Carl von Ossietzky Universität Oldenburg

Fachbereich Informatik

Studiengang Diplom-Informatik

Zwischenbericht Teil A

der Projektgruppe

Virtuelle naturwissenschaftlich-technische Labore im Internet

Thomas Aden, Jörg Appel, Wilko Heuten,

Stefan Koopmann, Tobias Krüger, Marco Lindner,

Mario Sachteleben, Holger Schneider, Christian Wichmann,

Dietrich Boles, Peter Dawabi, Marco Schlattmann

Oldenburg, Juni 1998

Inhaltsverzeichnis

1 Einleitung **1**

2 Anmerkungen zur Vortragsgestaltung **3**

2.1 Folien . . . 3

2.1.1 Drumherum . . . 3

2.1.2 Mittendrin . . . 3

2.1.3 Abbildungen . . . 4

2.2 Vortrag . . . 4

2.2.1 Struktur . . . 4

2.2.2 Vortragen . . . 4

2.2.3 Zwischenfragen . . . 5

2.2.4 Vorbereitung . . . 5

I Multimedia-Softwareentwicklung **7**

3 Multimedia-Software-Engineering **9**

3.1 Zusammenfassung . . . 9

3.2 Einleitung . . . 9

3.3 Software-Engineering . . . 10

3.3.1 Geschichte . . . 10

3.3.2 Softwarequalitätsmerkmale und ihre Bedeutung . . . 10

3.3.3 Phasen eines Softwareprojekts/Phasenmodelle . . . 12

3.4 Multimedia-Software-Engineering . . . 17

3.4.1 Charakteristische Merkmale der Multimedia-Softwareentwicklung . . . 17

3.4.2 Vorgehensweise bei der Erstellung von Multimedia-Softwareprodukten 17
3.5 Autorensysteme 20
3.5.1 Einordnung der Autorensysteme in den Software-Entwicklungsprozeß 20
3.5.2 Frame-basierte Autorensysteme 21
3.5.3 Timeline-basierte Autorensysteme 21
3.5.4 Flowchart-basierte Autorensysteme 22
3.5.5 Bewertung 23
3.6 Abschließende Anmerkungen 24

4 Projektmanagement und Organisation 25

4.1 Zusammenfassung 25
4.2 Einleitung 25
4.3 Planung 26
4.3.1 Aufbau von Projektplänen 26
4.3.2 Zeitplanung 29
4.3.3 Projektplanung 29
4.4 Organisation 30
4.4.1 Betriebsstruktur 30
4.4.2 Gestaltung der Aufbauorganisation 31
4.4.3 Projektstrukturen 32
4.5 Personal 34
4.5.1 Allgemeine Qualifikationen 34
4.5.2 Aufgaben und Aktivitäten 34
4.6 Leitung 36
4.6.1 Teamarbeit 36
4.6.2 Kreativität 37
4.7 Kontrolle 37
4.7.1 Aufgaben 37
4.7.2 Konfigurationsverwaltung 38
4.8 Abschließende Anmerkungen 38

5 OO Softwareentwicklung mit der UML **41**

5.1 Zusammenfassung 41

5.2 Einleitung 41

5.3 Grundbegriffe der Objektorientierung 42

5.3.1 Vererbung und abstrakte Klassen 43

5.3.2 Behälterklassen 44

5.3.3 Kommunikation zwischen Objekten 44

5.3.4 Polymorphie 46

5.4 Objektorientierte Softwareentwicklung 46

5.4.1 Das Vorgehensmodell 47

5.4.2 Analyse 48

5.4.3 Design 49

5.4.4 Implementierung 51

5.4.5 Der operative Einsatz 52

5.5 Die Unified Modeling Language (UML) 52

5.5.1 Die Diagrammtypen der UML 53

5.5.2 Extensionsmechanismen der UML 59

5.6 Abschließende Bemerkungen 59

II Virtual Reality **61**

6 3D - Modellierung **63**

6.1 Zusammenfassung 63

6.2 Motivation 63

6.3 Einsatz von 3D-Computergrafiken 63

6.3.1 Vorteile der Computergrafik gegenüber realen Modellen 64

6.3.2 Unterschiede zwischen CAD und 3D-Modellierung 64

6.4 2D-Modellierung 65

6.5 Modellierung von Festkörpern 65

6.6 Materialien 68

6.6.1 Farbe 68

6.6.2 Spiegelreflexionen . . . 68
6.6.3 Glanzreflektion . . . 68
6.6.4 Diffuse Reflexion . . . 69
6.6.5 Ambiente Reflexion . . . 69
6.6.6 Transparenz . . . 69
6.6.7 Brechung . . . 69
6.6.8 Glühen . . . 69
6.6.9 Texturen . . . 69
6.7 Licht und Beleuchtung . . . 70
6.7.1 Beleuchtungsarten . . . 71
6.8 Szene und Navigation . . . 72
6.8.1 Koordinatensysteme . . . 72
6.8.2 Kameras . . . 73
6.9 Rendering . . . 73
6.9.1 Wireframe (Drahtgitter) . . . 73
6.9.2 Hidden-line Rendering (Ermittlung sichtbarer Flächen) . . . 73
6.9.3 Flat Shading . . . 73
6.9.4 Gouraud Shading . . . 73
6.9.5 Phong Shading . . . 74
6.9.6 Raytracing . . . 74
6.9.7 Radiosity . . . 74
6.10 3D-Grafik Software . . . 75
6.10.1 Extreme 3D 2.0 (Macromedia) . . . 75
6.10.2 Highlight Professional 2.0 (Ingenieurbüro M. Rahlff) . . . 75
6.10.3 Ray Dream Studio 4.0 (Fractal Design) . . . 75
6.10.4 Reflections 4.0 (Oberland) . . . 75
6.10.5 Visual Reality 2.0 (Visual Software) . . . 76
6.10.6 3D Studio MAX (Autodesk) . . . 76
6.10.7 Cinema 4D (Maxon) . . . 76
6.10.8 Ray Dream Studio 5 (Fractal Design) . . . 77
6.10.9 Persistance of Vision (POVRay) . . . 77
6.11 Abschliessende Bemerkungen . . . 78

7 VRML - das Web in 3D **79**

7.1 Zusammenfassung 79

7.2 Einleitung 79

7.3 Geschichtliches 80

7.3.1 Die Anfänge 80

7.3.2 Der Weg zum ISO-Standard 81

7.3.3 Blick in die Zukunft 81

7.4 Möglichkeiten in VRML 82

7.4.1 Sichtbare Objekte 82

7.4.2 Licht und Schatten 83

7.4.3 Mehr reality 83

7.4.4 Navigation 84

7.5 Hinter den Kulissen 84

7.5.1 Aufbau von VRML-Dateien 84

7.5.2 Knotentypen 85

7.5.3 Aufbau von Knoten 86

7.5.4 Ereignisse und Routen 87

7.5.5 Mehrfachverwendung und Prototypen 87

7.5.6 Scripting 88

7.5.7 Transformationen 89

7.5.8 Eine Mini-Welt 89

7.6 VRML-Browsing 91

7.6.1 Varianten und Anforderungen 91

7.6.2 Cosmo Player 2.0 91

7.7 VRML-Authoring 92

7.7.1 Der Weg zur eigenen VRML-Welt 92

7.7.2 Superscape 3D Webmaster 92

7.7.3 RenderSoft VRML Editor 92

7.8 Abschließende Anmerkungen 93

8 QuickTime VR und das QTVR Authoring Studio 95

8.1 Zusammenfassung . . . 95
8.2 Einleitung . . . 95
8.3 QuickTime VR . . . 96
8.4 Das QuickTime VR Authoring Studio . . . 96
8.5 Erstellen von QTVR-Panoramen . . . 98
8.5.1 Erzeugen des Ausgangsmaterials . . . 98
8.5.2 Der Panorama Maker . . . 99
8.5.3 Der Panorama Stitcher . . . 100
8.6 Erstellen von QTVR-Objekten . . . 101
8.6.1 Erzeugen des Ausgangsmaterials . . . 101
8.6.2 Der Object Maker . . . 102
8.7 Erstellen von QTVR-Szenen . . . 103
8.7.1 Der Scene Maker . . . 103
8.7.2 Größere QTVR-Projekte verwalten . . . 104
8.8 Kompression . . . 104
8.9 Fazit . . . 104

III Elektronische Lehre 107

9 Rechnergestützte Kooperation 109

9.1 Zusammenfassung . . . 109
9.2 Rechnergestützte Kooperation im Kontext der Projektgruppe . . . 109
9.3 Universelles Modell rechnergestützter Kooperation . . . 110
9.3.1 Beispielszenario einer rechnergestützter Kooperation . . . 110
9.3.2 Die Elemente des Modells . . . 110
9.4 Kooperationsunterstützende Werkzeuge . . . 113
9.4.1 Kommunikationsaspekte kooperationsunterstützender Werkzeuge . . . 114
9.4.2 Koordinationsaspekte kooperationsunterstützender Werkzeuge . . . 115
9.4.3 Kooperationsaspekte kooperationsunterstützender Werkzeuge . . . 121
9.5 Anforderungen an ein universelles Werkzeug . . . 122

9.5.1 Anforderungen an die Kommunikationsunterstützung . . . 123
9.5.2 Anforderungen an die Koordinationsunterstützung . . . 123
9.5.3 Anforderungen an die Kooperationsunterstützung . . . 123
9.6 Fazit für die Projektgruppe . . . 123

10 Lehr-/Lernsysteme **125**
10.1 Zusammenfassung . . . 125
10.2 Einleitung . . . 125
10.3 Definition von Lernsoftware . . . 126
10.3.1 Multimedia . . . 126
10.3.2 Hypertext/Hypermedia . . . 126
10.3.3 Lernkonzepte . . . 127
10.3.4 Lernformen . . . 129
10.3.5 Begriffsvielfalt der Lernsoftware . . . 130
10.4 Gestaltungsmöglichkeiten für Lernsoftware . . . 131
10.4.1 Interaktion und Interaktivität . . . 131
10.4.2 Abbilder . . . 133
10.4.3 Hypertext und Hypermedia . . . 136
10.4.4 Kooperatives Lernen . . . 139
10.4.5 Adaptivität und Adaptierbarkeit . . . 142
10.5 Abschließende Anmerkungen . . . 144

11 Planungskonzepte am Beispiel von MOLGEN **145**
11.1 Zusammenfassung . . . 145
11.2 Einleitung . . . 145
11.3 Planungskonzepte . . . 147
11.3.1 System und Teilsystem . . . 147
11.3.2 Hierarchische Planung . . . 148
11.3.3 Constraints . . . 148
11.3.4 Meta-Planning . . . 149
11.4 Planungskonzepte am Beispiel von MOLGEN . . . 149
11.4.1 Planungsaufgabe . . . 150

11.4.2 Aufbau von MOLGEN 150
11.4.3 Das Experiment 152
11.5 Anwendungen im Rahmen der Projektgruppe 155
11.5.1 Simulationskomponente 156
11.5.2 Autorensystemkomponente 159
11.6 Abschließende Anmerkungen 159

Literatur **160**

Abbildungsverzeichnis

3.1 Softwarequalitätsmerkmale 11
3.2 Das klassische Software-Life-Cycle-Modell 13
3.3 Das Wasserfall-Modell 14
3.4 Das Prototyping-orientierte Life-Cycle-Modell 15
3.5 Das Spiral-Modell 16
3.6 Beispiel für ein „Flowchart“ 19
3.7 Anwendungsbereiche der Autorensysteme 20
3.8 Benutzungsoberfläche von HyperCard 22
3.9 Benutzeroberfläche von Macromedia Director 23
3.10 Benutzeroberfläche von Authorware Professional 24

4.1 Ein Beispiel für ein Projektplan 27
4.2 Netzplanarten im Vergleich 28
4.3 Netzplandarstellung 29
4.4 Die fünf Teile einer Organisation 31

5.1 Klasse und Objekt in der UML-Notation 42
5.2 Klassenhierarchie in der UML-Notation 43
5.3 Die Klasse *GUI::Container* und ihre Einordnung in die Hierarchie 44
5.4 Kompositionsbeziehungen 45
5.5 Nachrichten 46
5.6 4–Phasen–Zyklus 48
5.7 Analyse–Phase 48
5.8 Design–Phase 50
5.9 Implementierung 51

5.10 Operativer Einsatz . . . 52
5.11 Anwendungsfalldiagramm . . . 53
5.12 Einige Elemente des Klassendiagrammes . . . 54
5.13 Kollaborationsdiagramm . . . 55
5.14 Sequenzdiagramm . . . 56
5.15 Zustandsdiagramm . . . 57
5.16 Aktivitätsdiagramm . . . 57
5.17 Komponentendiagramm . . . 58
5.18 Einsatzdiagramm . . . 58

6.1 Kurvenarten: a) Hermite Kurve mit ihren Tangentenvektoren, b) Bezier-Kurve mit ihrer konvexen Hülle und c) uniforme B-Spline Kurve . . . 66
6.2 Boolesche Differenz zweier Quader . . . 66
6.3 Extrusion . . . 67
6.4 Sweeping . . . 67
6.5 Lofting . . . 67
6.6 Studio Beleuchtung . . . 71
6.7 Die 4T-Ansicht . . . 72
6.8 Funktionsweise des Raytracings . . . 74

7.1 Verknüpfung von HTML- und VRML-Dokumenten . . . 80
7.2 Verschiedene Arten von Lichtquellen . . . 83
7.3 Lage im Koordinatensystem . . . 90
7.4 Cosmo Player 2.0 (Screenshot) . . . 91
7.5 Superscape 3D Webmaster (Screenshot) . . . 93
7.6 RenderSoft VRML Editor (Screenshot) . . . 93

8.1 Ablauf beim Erstellen von QTVR-Multi-Media . . . 97
8.2 Zusammenfügen von Einzelbildern zu einem Panoramabild . . . 100
8.3 Kompressionsverfahren im Überblick . . . 105

9.1 Kommunikationsmechanismus . . . 113
9.2 Morphologischer Kasten: Kommunikation . . . 115

9.3 Einordnung von Kommunikationswerkzeugen 116
9.4 Beispiel für eine Aktivitätenplanung 117
9.5 Morphologischer Kasten: Koordination 121
9.6 Einordnung von Koordinationswerkzeugen 122

10.1 Zeigefunktion 134
10.2 Situierungsfunktion 134
10.3 Konstruktionsfunktion 135

11.1 Beispiel eines Plans für den Erwerb eines Seminarscheins 146
11.2 MOLGENs drei Ebenen - laboratory, design und strategy space 150
11.3 Laboratory Space - abstrakte und spezifische Operatoren 151
11.4 Least-commitment und heuristic cycle im strategy space von MOLGEN 152
11.5 Schema von MOLGENs Schritten, um den ersten Operator zu finden 153
11.6 Zwischenergebnis MOLGENs - ein noch abstrakter Plan 154
11.7 Verfeinerung des Plans - die erste Bedingung (constraint) 155
11.8 Das Simulationsergebnis ergibt eine Abweichung zum Ziel 156
11.9 Propagating contraints - schematische Darstellung 157
11.10 Ansätze für die Simulation eines Labors - Zwischen Freiheit und Vorgabe, abstrakter Planung und Details 158

Kapitel 1

Einleitung

Dieser Bericht faßt die Ergebnisse zusammen, die etwa bei Halbzeit der Projektgruppe ELVIS (Virtuelle naturwissenschaftliche Labore im Internet) vorliegen. ELVIS ist eine Projektgruppe des Fachbereichs Informatik der Universität Oldenburg, die im SS 98 und im WS 98/99 stattfindet. Dieser Zwischenbericht setzt sich aus zwei Teilen zusammen:

- Teil A faßt den bisherigen Ablauf und die bisherigen Ergebnisse der Projektgruppe zusammen.
- Teil B enthält die ausgearbeiteten Seminarvorträge aus dem ersten Abschnitt der Projektgruppe.

Nach Abschluß der Projektgruppe im Frühjahr 1999 wird ein Endbericht erscheinen.

Innerhalb der Seminarphase der Projektgruppe wurden insgesamt 9 Vorträge gehalten, die die Teilnehmer der Projektgruppe organisatorisch und inhaltlich auf die Projektgruppe vorbereiten sollten. Die schriftlichen Ausarbeitungen der Seminarvorträge werden in diesem Bericht zusammengefaßt. Die Beiträge lassen sich dabei in drei Gruppen einteilen:

- Im Teil *Multimedia-Softwareentwicklung* wurden die drei Vorträge
  - Multimedia-Software-Engineering (Stefan Koopmann)
  - Projektmanagementg (Mario Sachteleben)
  - Objektorientierte Softwareentiwcklung (Thomas Aden)

  gehalten.
- Im Teil *Virtual Reality* wurden die drei Vorträge
  - 3D-Modellierung (Wilko Heuten)
  - VRML (Christian Wichmann)
  - Quicktime VR (Jörg Appel)

  gehalten.
- Im Teil *Elektronische Lehre* wurden die drei Vorträge

- Groupware (Holger Schneider)
- Lehr-/Lernsysteme (Marco Lindner)
- MolGen (Tobias Krüger)

gehalten.

Kapitel 2

Anmerkungen zur Vortragsgestaltung

2.1 Folien

2.1.1 Drumherum

- Titelblatt mit Vortragsthema, Referent, Datum, Veranstaltung
- Folie mit Gliederung des Vortrags
- einheitliches Layout
- evtl. Rahmen, zumindest Linien zur Gliederung der Folie
- Kopfzeile (oder Fußzeile) mit Thema des Vortrags, Referent
- für jede Folie eine Überschrift
- Foliennumerierung und Gesamtzahl der Folien angeben (Folie 5/20)
- Querformat
- keine bunt-überladenden Folien (keine Farben, wo es nicht sinnvoll ist)
- Orientierungshilfen (Überschrift, Gliederungspunkt)
- auf Rechtschreibung achten!

2.1.2 Mittendrin

- Schriftgröße mindestens 14-18 Punkt
- Standard-Schriftart (Times New Roman oder Arial) verwenden
- bei Unterpunkten kleinere Schriftgröße wahlen und einrücken
- maximal fünf bis sieben Items pro Folie, maximal 13 Zeilen
- ein Thema auf maximal zwei bis drei Folien verteilen

- pro Folie nur ein Thema, neues Thema, neue Folie
- Abbildungen verwenden (denn: tausend Bilder sagen mehr als ein Wort)
- nur Kernaussagen, Stichworte, keinen fortlaufenden Text verwenden
- wichtige Informationen in Folie aufnehmen, keine Details

2.1.3 Abbildungen

- Farben gezielt einsetzen
- Schrift immer groß genug wählen, zu kleine Schrift bringt dem Zuhörer nichts
- Graphiken, die aus Büchern übernommen werden, ggf. vergrößern bzw. für den Vortrag abändern, vereinfachen (Details weglassen, etc.)

2.2 Vortrag

2.2.1 Struktur

- Thema vorstellen, sich selbst vorstellen, Ziel vorstellen
- Gliederung erläutern
- roter Faden sollte erkennbar sein
- Gliederung evtl. immer wieder auflegen, um Orientierung zu ermöglichen
- Beispiele bringen (gut: durchgehendes Beispiel)
- Einleitung in das Thema
- Abschließende Bemerkungen, Fazit
- Zusammenhänge sollten klar werden

2.2.2 Vortragen

- nicht ablesen, sondern frei sprechen (erleichtert dem Zuhörer das Folgen)
- Bezug zur Folie halten, aber nicht Punkte einfach ablesen
- Publikum mit einbeziehen
- Blickkontakt halten (Reaktionen erkennen)
- Zeit einhalten, Zeit während des Vortrags im Auge haben
- Spannungsbogen (auf wichtige Aussagen hinweisen)

- langsame Sprechweise, aber nicht monoton
- zu neuer Folie hinführen (evtl. durch einleitende Frage, deren Antwort auf Folie steht)
- keine Gegenstände in die Hand nehmen, die Zuhörer ablenken können (der dauernd klickende Kugelschreiber)
- Medienwechsel (auch Tafel benutzen, Graphiken entwickeln)
- mit Stift o.a. zeigen, wo man sich auf der Folie befindet (Orientierungshilfe)
- Folien möglichst nicht abdecken
- Notizzettel nur benutzen, wenn man nicht ablesen oder sie sich erst durchlesen muß (siehe auch Ilona Christen)

2.2.3 Zwischenfragen

- vor Beginn klarstellen, wann man Fragen zuläßt (nur am Ende, zwischendrin, nur wenn man es fragt)
- Fragen kann man auch zurückstellen
- auf Fragen präzise antworten, nicht schwafeln
- nicht aggressiv werden, immer ruhig und souverän wirken, einfach so wirken, als ob es die leichteste Sache der Welt sei und man das schon tausendmal gemacht hätte, Verkäufertyp sein, lächeln!

2.2.4 Vorbereitung

- Vortrag zu Hause durchsprechen, üben, möglichst mit Zuhörer, der Kritik übt
- Vortrag auf Zeit abstimmen
- besser weniger Zeit als mehr Zeit benötigen
- flexibel auf Zeitvorgabe reagieren (Wo kann ich Vortrag abkürzen, unwichtigere Aussagen von wichtigen unterscheiden)
- Zielpublikum beachten (Vorkenntnisse, Interesse, Anwendung des Inhalts durch Zuhörer, Welche Begriffe müssen erklärt werden)
- Informationsgehalt der Aussagen überprüfen

Teil I

Multimedia-Softwareentwicklung

Kapitel 3

Multimedia-Software-Engineering

3.1 Zusammenfassung

Autor: Stefan Koopmann

Dieser Artikel beschäftigt sich zunächst mit den Grundlagen des Software-Engineerings. Dabei werden Qualitätsmerkmale von Software und die gängigsten Phasenmodelle zur Softwareentwicklung behandelt. Anschließend wird auf die charakteristischen Merkmale der Multimedia-Softwareentwicklung eingegangen und eine mögliche Vorgehensweise zur Herstellung von Multimedia-Software vorgestellt. Im letzten Kapitel geht es um die Einführung verschiedener Multimedia-Autorensysteme, die in der letzten Zeit stark an Bedeutung zugenommen haben.

3.2 Einleitung

Der Begriff „Multimedia“ ist seit ein paar Jahren in aller Munde. Begründet ist dies durch die neuen Möglichkeiten, die „Multimedia“ im Bereich des elektronischen Publizierens mit sich bringt. Informationen können mit Bildern, Audios oder Videos angereichert und so besser vermittelt werden. Neben den offensichtlichen Vorteilen von Multimedia ist allerdings auch ein gewisses Umdenken in der Erstellung dieser Softwareprodukte notwendig. Die seit den 60er Jahren entwickelten klassischen Phasenmodelle lassen aufgrund der besonderen Charakteristiken der Multimedia-Software-Produktion nicht ohne weiteres auf die Herstellung von Multimediaanwendungen übertragen. Zur Unterstützung der Erstellung multimedialer Softwareprodukte sind daher in den letzten Jahren spezielle Autorensysteme entwickelt worden. Sie sollen auch Personen ohne Programmierkenntnissen das elektronische Publizieren mit Multimedia-Elementen ermöglichen. Diese Arbeit gibt einen Einblick in die Besonderheiten der Entwicklung von Multimedia-Softwareprodukten. Zudem werden die wichtigsten Klassen von Autorensystemen vorgestellt.

3.3 Software-Engineering

3.3.1 Geschichte

Die Aufgabe der ersten Programmierer bestand zumeist darin, bereits bekannte Algorithmen in eine Programmiersprache umzusetzen. Die damaligen Probleme waren noch verhältnismäßig einfach, und der Programmierer war auch meistens der einzige Benutzer des Programms. Probleme traten auf, als man begann, komplexe wissenschaftliche, technische und kommerzielle Probleme mit Hilfe des Computers zu lösen. Von nun an arbeiteten mehrere Personen an einem Programm, und es ergaben sich Schwierigkeiten wie z.B. die sinnvolle Aufgabenverteilung oder die Spezifikation der Schnittstellen. Da die Programmierer diesen Aufgaben nicht gewachsen waren, begann man um 1965 von der sogenannten „Softwarekrise“ zu sprechen. Nachdem zuvor die Hardware keine komplexen Programme zugelassen hatte, wurde jetzt also die Softwareentwicklung zum „begrenzenden Faktor“ in der Computertechnologie. Folglich rückte die Verbesserung der Programmiertechnologie immer mehr in den Vordergrund des Interesses. Auf ersten Software-Engineering-Konferenzen Ende der 60er-Jahre forderte man daher eine Wandlung von der „Kunst des Programmierens“ hin zu einer ingenieurmäßigen Softwareherstellung. Der Begriff „Software-Engineering“ wird von Sommerville wie folgt definiert: „Software Engineering befaßt sich mit dem Bau von Softwaresystemen, die nicht von einem Entwickler allein hergestellt werden können. Software Engineering beruht auf der Anwendung ingenieurmäßiger Prinzipien und umfaßt sowohl technische als auch nichttechnische Aspekte.“ [Pom93]

3.3.2 Softwarequalitätsmerkmale und ihre Bedeutung

Softwareprodukte sollen also durchaus nach ingenieurmäßigen Gesichtspunkten hergestellt werden. Nun liegt die Frage nahe, ob sich die Qualität von Softwareprodukten ebenso messen läßt, wie die Qualität anderer technischer Produkte. Das ist leider nicht der Fall. Trotzdem ist sich die Fachwelt weitgehend einig, nach welchen Merkmalen die Qualität von Softwareprodukten beurteilt werden kann (vgl. Abbildung 3.1).

Ein Programm ist *korrekt*, wenn der Programmtext mit der Spezifikation des Programms übereinstimmt. Die Korrektheit spielt besonders da eine wichtige Rolle, wo Programme in großen Programmsystemen eingebettet sind, da ein nicht korrektes Programm auch zu einem nicht korrektem Programmsystem führt. Ein Programm kann als *zuverlässig* bezeichnet werden, wenn es mit einer hohen Wahrscheinlichkeit innerhalb eines vorgegebenen Zeitintervalls für bestimmte Eingabewerte die gewünschten Ausgaben liefert. Unter *Benutzerfreundlichkeit* werden die Begriffe Adäquatheit, Erlernbarkeit und Robustheit zusammengefaßt. Mit *Adäquatheit* ist gemeint, daß

- die vom Benutzer geforderten Eingaben möglichst minimal sein sollten. Des weiteren sollte eine flexible Dateneingabe ermöglicht werden.
- die Funktionalität des Programms auf die in der Spezifikation festgelegten Kriterien beschränkt bleiben sollte. Dabei sollte allerdings die Erweiterbarkeit trotzdem gegeben sein.
- die Ausgabe der Ergebnisse gut strukturiert und für den Benutzer einfach interpretierbar ist. Außerdem soll dem Benutzer die Möglichkeit gegeben werden, den Umfang, den Detailliertheitsgrad und die Art der Präsentation der Ausgaben zu bestimmen.

Abbildung 3.1: Softwarequalitätsmerkmale

Bei der *Erlernbarkeit* eines Programms kommt es vor allem auf die Ausgestaltung der Benutzerschnittstellen an. Weiterhin ist ein klares, gut strukturiertes Benutzerhandbuch von Vorteil. Mit *Robustheit* assoziiert man eine gewisse Fehlertoleranz des Programms. Dabei sollten besonders häufig zu erwartende Fehler eine angemessene Reaktion des Programms hervorrufen. Das Auffinden von Programmfehlern, die Durchführung von Fehlerkorrekturen, sowie die Eignung zur Erweiterbarkeit eines Programms faßt man unter dem Begriff *Wartungsfreundlichkeit* zusammen. Folgende Kriterien begünstigen die Wartungsfreundlichkeit eines Programms:

- Ein guter Programmierstil, die Strukturiertheit des Systems und eine gute Dokumentation sorgen für eine angemessene *Lesbarkeit* des Programms.
- Wenn sich zu einem Programm Änderungen einfach und ohne unerwünschte Nebenwirkungen hinzufügen lassen, spricht man von einer guten *Erweiterbarkeit*.
- Ein Programm ist gut *testbar*, wenn Programmabläufe leicht verfolgt und Fehler somit schnell lokalisiert werden können.

Ein Programm kann als *effizient* betrachtet werden, wenn Ressourcen, wie Zeit, Speicherplatz, Übertragungskanäle und periphere Einheiten optimal genutzt werden. Ein Programm besitzt eine große *Portabilität*, wenn es ohne größere Schwierigkeiten auf anderen Rechnern ausgeführt werden kann. Man spricht dabei auch von dem Grad der Rechnerunabhängigkeit.

Leider können bei der Softwareentwicklung nicht alle Qualitätskriterien gleichermaßen berücksichtigt werden, da sich eine starke Berücksichtigung des einen unter Umständen negativ auf ein anderes

Kriterium auswirkt. So bewirkt z.B. eine gute Erweiterbarkeit des Programms in der Regel eine schlechtere Effizienz. Auf der anderen Seite können sich die Faktoren auch positiv beeinflussen. So hat z.B. die Robustheit eines Programms im allgemeinen auch eine bessere Testbarkeit zur Folge.

3.3.3 Phasen eines Softwareprojekts/Phasenmodelle

Um ein Softwareprojekt besser durchführbar und planbarer zu machen, begann man in den 60er Jahren Vorgehensmodelle zu entwickeln. Diese Modelle sollen den Ablauf des Lösungsprozesses regeln und ihn in überschaubare Abschnitte einteilen. Die einzelnen Phasen der Modelle und ihre zeitliche Beziehung zueinander bezeichnet man auch als *Software-Life-Cycle*. Typischerwiese gliedert man den Software-Entwicklungsprozeß in folgende Phasen:

- Problemanalyse und Planung
- Systemspezifikation(Anforderungsdefinition)
- System- und Komponentenentwurf
- Implementierung und Komponententest
- Systemtest
- Betrieb und Wartung

Ursprünglich ging man davon aus, daß die einzelnen Phasen getrennt voneinander zu betrachten seien. Erst wenn eine Phase mit dem entsprechendem Ergebnis beendet war, sollte die nächste Phase in Angriff genommen werden. In der Realität zeigte sich jedoch bald, daß ein solches streng sequentielles Vorgehen nur in den seltensten Fällen durchführbar ist. Nachfolgend wird nun auf die aus heutiger Sicht wichtigsten Phasenmodelle eingegangen.

Das klassische Software-Life-Cycle-Modell

Beim klassischen Software-Life-Cycle-Modell(vgl. Abbildung 3.2) unterscheidet man folgende Phasen:

- Problemanalyse und Planung

  In der Problemanalyse-Phase soll zunächst einmal der Ist-Zustand analysiert werden. Es muß festgestellt werden, welche Arbeitsschritte durchgeführt werden müssen und welche Ressourcen dafür benötigt werden. Dabei ist zu beachten, daß die benötigten technischen, personellen, finanziellen und zeitlichen Ressourcen nicht die tatsächlich verfügbaren Ressourcen übersteigen. Außerdem soll bereits ein Projektplan erstellt werden.

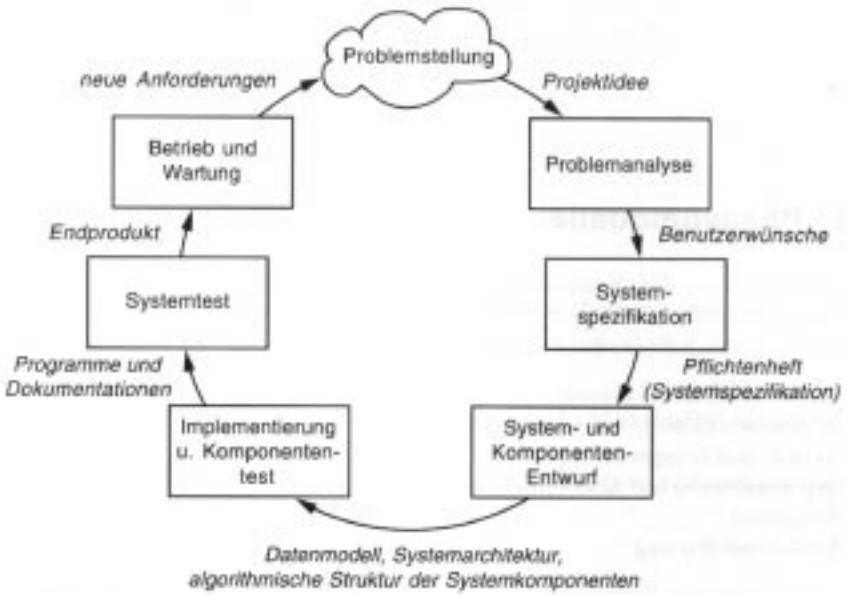


Abbildung 3.2: Das klassische Software-Life-Cycle-Modell

- Systemspezifikation

  Die wichtigsten Aktivitäten der Spezifikationsphase sind die Erstellung der Anforderungsdefinition sowie deren Prüfung bezüglich Vollständigkeit und Durchführbarkeit und die Erstellung eines genauen Projektplans.

- System- und Komponentenentwurf

  In dieser Phase werden die Systemkomponenten definiert und die Schnittstellen entworfen. Als Ergebnis sollte dann ein logisches Datenmodell der Systemarchitektur, die algorithmische Struktur der Systemkomponenten sowie die Dokumentation der Entwurfsentscheidungen vorliegen.

- Implementierung und Komponententest

  In der Implementierungsphase werden die Algorithmen verfeinert und in eine Programmiersprache umgesetzt. Außerdem erfolgen die Übertragung des logischen Datenmodells in ein physisches Datenmodell(z.B. Datenbank) und abschließend die Komponententests.

- Systemtest

  Die Wechselwirkungen der Systemkomponenten werden unter realen Bedingungen getestet. Dabei sollen möglichst viele Fehler des Systems aufgedeckt werden.

- Betrieb und Wartung

  Nachdem das System zur Benutzung freigegeben worden ist, müssen auftretende Fehler beseitigt werden. Außerdem werden Systemänderungen bzw. -erweiterungen durchgeführt.

Charakteristisch für die life-cycle-orientierte Software-Entwicklung ist das Prinzip der „Top-Down-Dekomposition“. Ausgehend von der Problemstellung erfolgt eine schrittweise Konkretisierung der Lösung

bis hin zur Implementierung. Als Nachteil dieses Modells hat sich die starre, sequentielle Ausführung der einzelnen Phasen erwiesen. Auf der anderen Seite liefert es einen klaren Rahmen des Projekts, so daß dessen Planung erleichtert wird. Aufgrund dieser Unzulänglichkeiten wurden weitere Modelle für die Softwareentwicklung vorgeschlagen.

Das Wasserfall-Modell

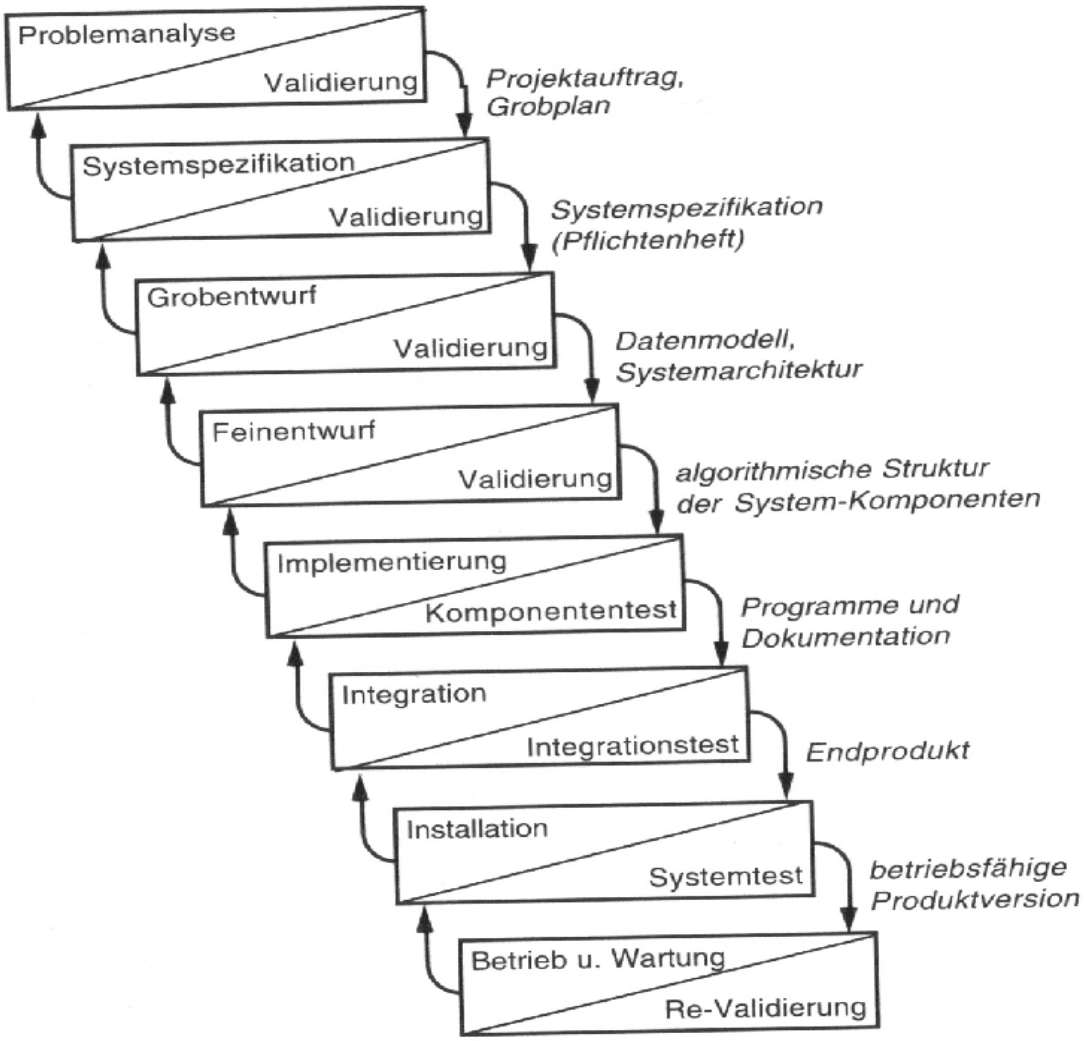


Abbildung 3.3: Das Wasserfall-Modell

Im Vergleich zum sequentiellen Modell wurden beim Wasserfall-Modell(vgl. Abbildung 3.3) Rückkopplungen zwischen den einzelnen Phasen hinzugefügt. Diese Rückkopplungen sollten aber nur zwischen zwei benachbarten Phasen stattfinden, um hohe Kosten bei der Fehlerbeseitigung zu vermeiden. Außerdem ist eine Validierung der Ergebnisse der einzelnen Phasen vorgesehen, die möglichst experimentell erfolgen soll.

Das Prototyping-orientierte Life-Cycle-Modell

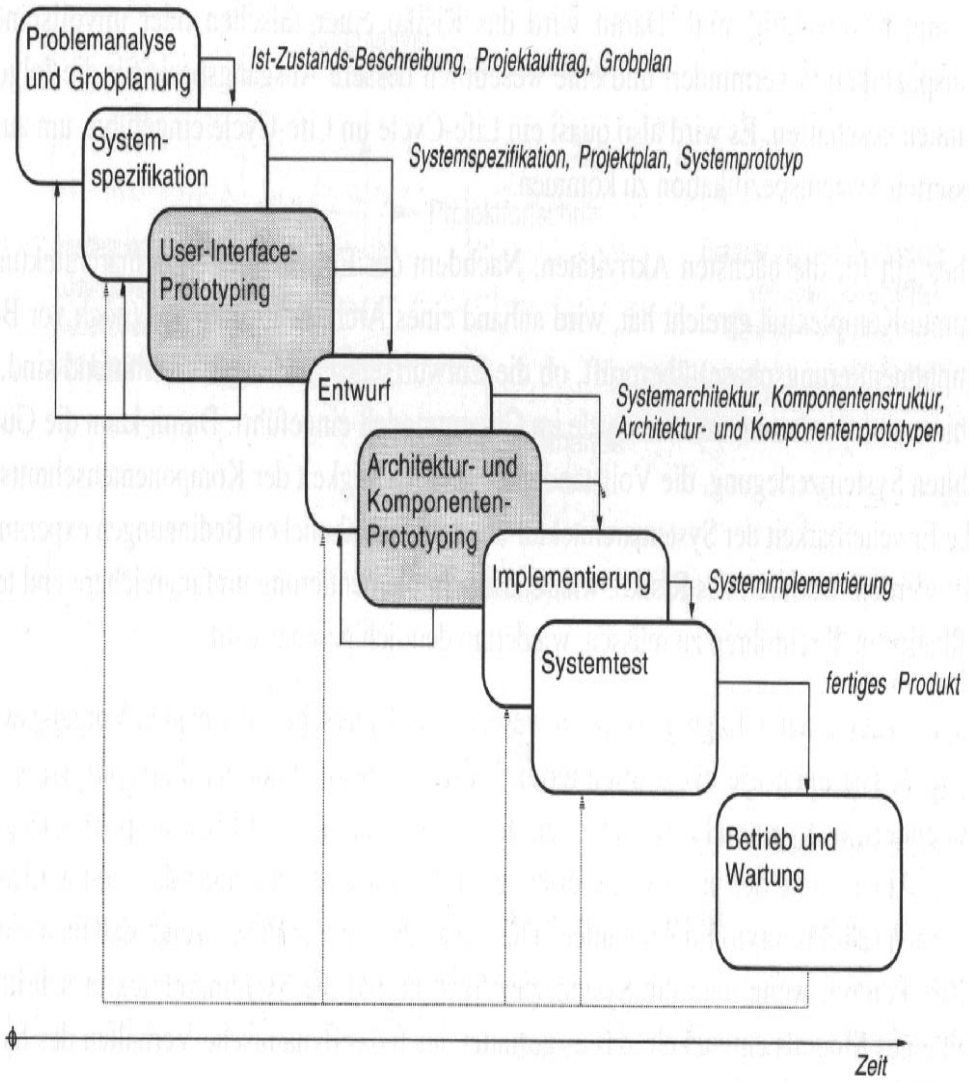


Abbildung 3.4: Das Prototyping-orientierte Life-Cycle-Modell

Dieses Prototyping-orientierte Modell (vgl. Abbildung 3.4) ist nicht als sequentielles, sondern als iteratives Modell anzusehen. Als Zwischenergebnisse werden nämlich Prototypen erstellt, die gegebenenfalls wieder überarbeitet werden. Der Vorteil des Prototypings liegt darin, daß Auftraggeber und Entwickler in der Lage sind, sich schon zu einem frühen Zeitpunkt des Projekts ein Bild zu machen , wie später das Endprodukt aussehen wird. Es können dann, falls notwendig, entsprechende Änderungen vorgenommen werden. Während man beim klassischen Life-Cycle-Modell also so spät wie möglich mit der Implementierung beginnt, erfolgt diese hier schon zu einem möglichst frühen Zeitpunkt. Die einzelnen Phasen unterscheiden sich zwar geringfügig von denen der vorher behandelten Modelle, jedoch fällt bei diesem Ansatz die deutliche zeitliche Überlappung der einzelnen Phasen auf.

Das Spiral-Modell

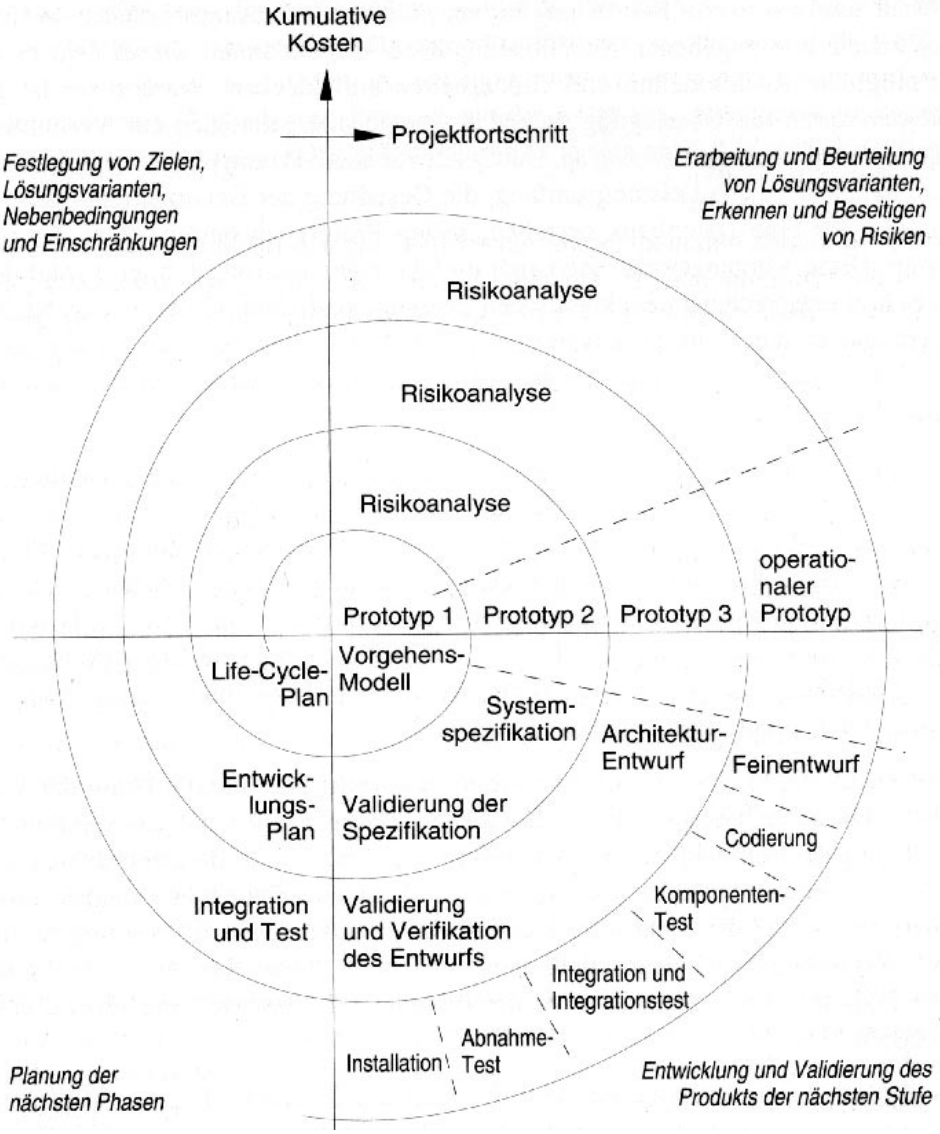


Abbildung 3.5: Das Spiral-Modell

Das Spiral-Modell (vgl. Abbildung 3.5) stellt eine Kombination bereits behandelter Vorgehensmodelle dar. Die Radien geben den bis zu einem bestimmten Zeitpunkt vollbrachten Gesamtaufwand an, während die Winkel den Projektfortschritt innerhalb der einzelnen Phasen, die immer in der Fertigstellung eines Prototyps münden, anzeigen. Jeder dieser Spiralenzyklen beginnt mit der Festlegung der Anforderungen, denen das (Zwischen-)Produkt genügen muß, sowie mit Überlegungen, wie dieses Ziel am besten erreicht werden kann. Anschließend sollen Risikoquellen und Unklarheiten der verschiedenen Lösungsansätze gefunden und mit Hilfe von Prototyping behandelt werden. In der Validierungsphase setzen sich alle beteiligten Personen zusammen und besprechen das in dem jeweiligem Zyklus Erreichte sowie die Ziele für den nächsten Zyklus. Dieser Zyklus setzt sich dann in dieser Form weiter fort. Dieses Modell ist sowohl auf die Erstellung, als auch auf die Wartung von Software anwendbar.

3.4 Multimedia-Software-Engineering

3.4.1 Charakteristische Merkmale der Multimedia-Softwareentwicklung

Die beschriebenen Modelle zur Softwareentwicklung lassen sich nach [Bol97a] jedoch nicht ohne weiteres für die Erstellung von Multimedia-Produkten verwenden. Beispielsweise ist ein Umdenken in konzeptioneller Sicht notwendig. Fragen, die sich mit der Auswahl von Medientypen, den Interaktionsmöglichkeiten oder der Layoutgestaltung befassen, spielen bei der „normalen“ Softwareentwicklung eine weniger bedeutende Rolle. Auch kommen technische Aspekte, die mit den neuen Medien unmittelbar verknüpft sind, wie Datenkompression oder Synchronisation, hinzu. Umfragen haben gezeigt, daß die klassischen Modelle in der Multimedia-Entwicklung kaum zum Einsatz kommen. Die Unternehmen selbst bezeichnen ihre Vorgehensweise bei der Softwareentwicklung als eher *ad hoc*. So ist zu befürchten, daß analog zu dem Begriff *Softwarekrise* aus den 60er Jahren in näherer Zukunft eine *Multimedia-Krise* eintreten könnte. Folgende Punkte aus [Bol97b] sind die wichtigsten Merkmale der Entwicklung von Multimedia-Software im Vergleich zur gewöhnlichen Softwareentwicklung:

- An einer Multmedia-Produktion sind häufig Personen verschiedenster Qualifikation beteiligt(z.B. Informatiker, Graphik-Designer, Medien-Spezialisten, ...).
- Neben den rein programmiertechnischen Tätigkeiten müssen auch künstlerische, ästhetische und psychologische Aspekte berücksichtigt werden.
- Medienobjekte wie Animationen, Videos oder Audios müssen erstellt und in das System integriert werden.

Beim Versuch, die klassischen Phasenmodelle auf die Entwicklung von Multimedia-Produktionen zu übertragen, kommt man zu dem Ergebnis, daß das reine Software-Life-Cycle-Modell wegen der Bedeutung der Benutzerschnittstellengestaltung nicht anwendbar ist. Auch die Verwendung des prototypischen Ansatzes wird wegen eventueller Einbeziehung von Fremdfirmen als wenig sinnvoll angesehen. Da die Risikoanalysen in jedem Zyklus des Spiralmodells als zu aufwendig angesehen werden, um eine wirtschaftliche Produktion zu gewährleisten, wird auch dieses Modell abgelehnt. Einige Autoren haben deshalb eigene Modelle entwickelt, die auf den klassischen Modellen basieren, dabei aber auf die speziellen Anforderungen der Multmedia-Produktion eingehen [Bol97a]].

3.4.2 Vorgehensweise bei der Erstellung von Multimedia-Softwareprodukten

Nachfolgend wird eine mögliche Vorgehensweise für die Erstellung von Multimedia-Softwareprodukten vorgestellt. Der Entwicklungsprozeß wurde dabei in drei Phasen(*Pre-Production, Production, Post-Production*) gegliedert. Hergeleitet wurde dieser Ansatz aus [Bur93] und [Su95].

- **„Pre-Production“**

  Zunächst einmal müssen sich Auftraggeber und Softwareentwickler zusammensetzen, um die Ziele des Projekts schriftlich zu fixieren. An diesen ersten Sitzungen nehmen etwa vier bis acht Personen teil, je die Hälfte von Auftraggeber- und Entwicklerseite. Der Umfang des Schriftstückes kann von ein paar Schlagzeilen bis hin zu mehreren Seiten Text sehr verschieden sein. Entscheidend ist auf jeden Fall, daß sich beide Seiten über die Ziele des Projekts klar werden, bevor mit

der eigentlichen Arbeit begonnen wird. Dabei müssen neben den rein inhaltlichen Aspekten auch Punkte wie die Zielgruppe und die Einsatzumgebung des Produkts behandelt werden. Eventuell sollten auch Interviews mit den zukünftigen Anwendern geführt werden. Nachdem Menge und Art der bevorstehenden Arbeit bekannt sind, muß der Projektleiter sein Team zusammenstellen. Er muß dabei berücksichtigen, wieviele Personen er benötigt, aus welchen Fachrichtungen diese Personen kommen sollen (Programmierer, Graphiker, Werbefachleute, ...) und ob auch außenstehende Firmen mit Aufgaben beauftragt werden sollen. Um Unstimmigkeiten gar nicht erst aufkommen zu lassen, muß der Projektleiter die Art der Arbeit für die einzelnen Mitarbeiter definieren und die Kompetenzen klar verteilen. Außerdem ist er für die Kontrolle der Arbeitsabläufe zuständig. Nachfolgend sollte der Markt auf Produkte untersucht werden, die dem zu erstellendem ähnlich sind. Das dient dazu, den Bedarf des Produktes festzustellen und auch Ideen für das eigene Produkt zu sammeln. Als nächstes wird der Zeitplan des Projekts abgesteckt. Der Projektleiter sollte bei der Erstellung des internen „Meilensteinplans“ darauf achten, daß dieser nicht zu eng gestaltet wird, da es immer wieder zu unerwarteten Zwischenfällen kommen kann, die eine Einhaltung des Plans dann unmöglich machen.

- **„Production“**

Es folgt ein „Brainstorming“ aller am Projekt beteiligten Personen. Es ist aber auch durchaus empfehlenswert, außenstehende Personen mit einzubeziehen, da diese in ihrer Denkweise oft weniger eingeengt sind und unkonventionelle Ideen einbringen können. Insgesamt gilt, daß alle Ideen vorgetragen werden sollten, auch wenn sie auf den ersten Blick nicht realisierbar zu sein scheinen oder mit den Wünschen des Kunden nicht konform sind. Anschließend werden die guten Ideen herausgefiltert, aufbereitet und dem Auftraggeber vorgelegt. Sobald der Auftraggeber sich für ein Modell entschieden hat, muß die für beide Seiten später verbindliche Anforderungsdefinition erstellt werden. In ihr enthalten sind Aussagen darüber, was das System leisten soll, nicht aber, wie dies zu realisieren ist. In der Entwurfsphase werden nun die Inhalte strukturiert. Dazu muß festgelegt werden, welche Informationen dargestellt werden und mit Hilfe welcher Medien sie präsentiert werden sollen. Ebenfalls festzulegen sind die Medien, die zur Gestaltung der Seiten verwendet werden sollen. Der „Look-and-feel“-Charakter einer Produktion spielt ebenso eine wichtige Rolle, wie die einheitliche Gestaltung der einzelnen Seiten. Weiterhin sind didaktische Überlegungen anzustellen. Dabei spielen Fragen eine Rolle wie z.B.: „Soll der Inhalt vom Benutzer aktiv erarbeitet werden oder erfolgt die Vermittlung der Informationen passiv?“ oder „Soll das Programm für den einzelnen Benutzer konfigurierbar sein?“. Auch ist klarzustellen, wie groß der Grad der Interaktion für den Benutzer sein soll und in welcher Form die Interaktion erfolgen soll. Am Ende der Entwurfsphase schließlich soll ein Drehbuch(*Flowchart*) stehen, das die Struktur des Systems veranschaulicht.

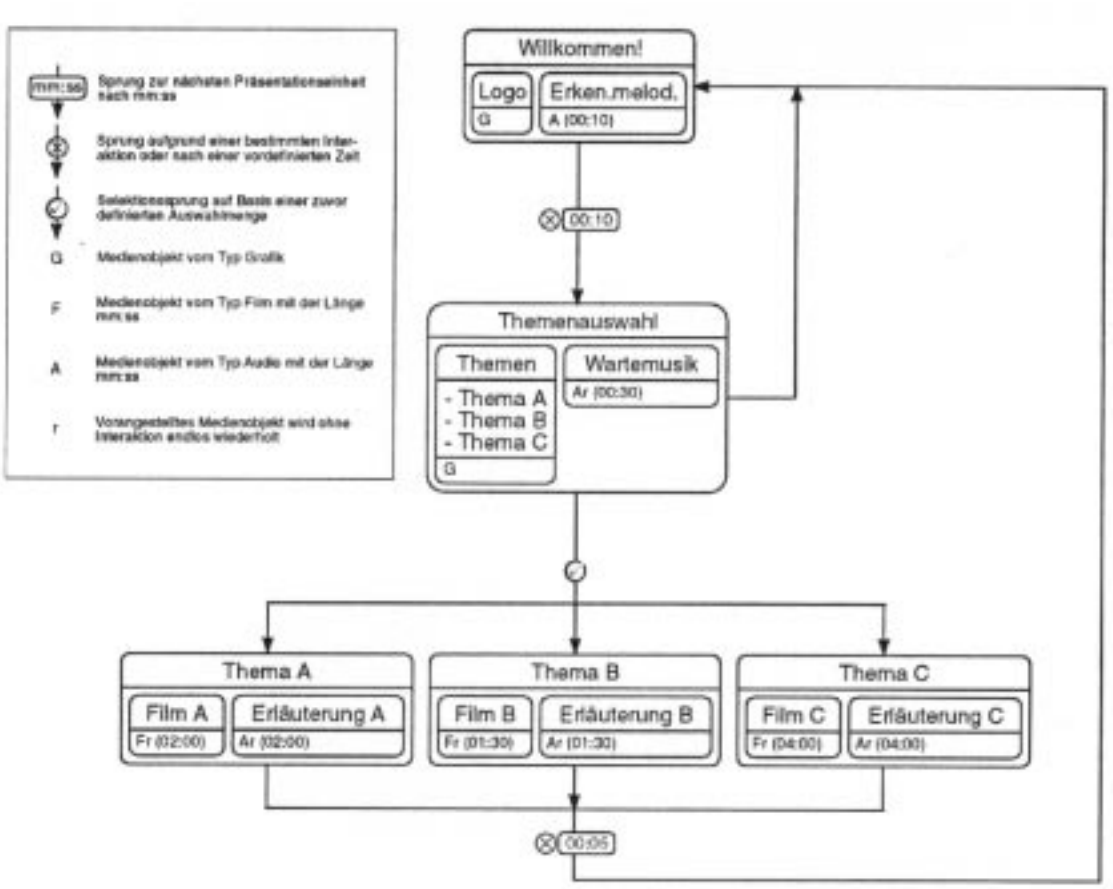


Abbildung 3.6: Beispiel für ein „Flowchart“

Die Abbildung 3.6 zeigt ein sehr einfaches Beispiel für ein Flowchart und ist aus [Dau95] entnommen. Eine charakteristische Tätigkeit beim Erstellen von Multimedia-Software ist die Medienerstellung bzw. Medienakquisition. Dabei ist natürlich zunächst zu überlegen, was für Videos, Audios, usw. in das Softwareprodukt integriert werden sollen. Jetzt kann man sich entweder dafür entscheiden, bereits vorhandene Medien zu nutzen, Medien selbst zu erstellen oder die Erstellung bei einer Fremdfirma in Auftrag zu geben. Dabei sollte auch eine Kostenabschätzung erfolgen. Bei der Suche nach bereits vorhandenen Fotos können z.B. Bildarchive im Internet sehr nützlich sein. Dabei sind aber immer die Copyright-Rechte zu beachten. Parallel zur Medienerstellung bzw. Medienakquisition kann bereits die Implementierung erfolgen. Tests der einzelnen Systemkomponenten werden bereits während der Implementierungsphase durchgeführt. Abschließend erfolgt dann der Test des Gesamtsystems. Auftretende Fehler sind ausführlich zu dokumentieren(eingesetzter Rechner, Fehlerquelle, Problembeschreibung, ...), damit der Fehler später besser lokalisiert werden kann. Bei den Test unterscheidet man zwischen den technischen Tests und den Verständnistests. Die technischen Tests sollten von Personen durchgeführt werden, die bereits Erfahrung mit Multimedia-Systemen besitzen. Bei den Verständnistests sollte man auch unerprobte Benutzer hinzuziehen, da gerade das „intuitive Benutzen“ ein Multimedia-System ausmacht. Außerdem gilt, was bei Tests aller Softwareprodukte gilt: Die Tests sollte niemals vom Programmierer selbst durchgeführt werde, da diesem der nötige Abstand zum Produkt fehlt.

- In der „**Post-Production**“ schließlich wird das fertige System beim Kunden installiert. Neben der

Konfiguration der Hard- und Software fällt auch die Wartung mit der Beseitigung von im realen Betrieb aufgetretenen Fehlern in diesen Bereich. Um festzustellen, ob das System letztendlich einen Zweck erfüllt, können Evaluationen durchgeführt werden.

3.5 Autorensysteme

3.5.1 Einordnung der Autorensysteme in den Software-Entwicklungsprozeß

Der technische Entwicklungsprozeß einer multimedialen Anwendung läßt sich in mehrere Phasen gliedern (vgl. Abbildung 3.7).

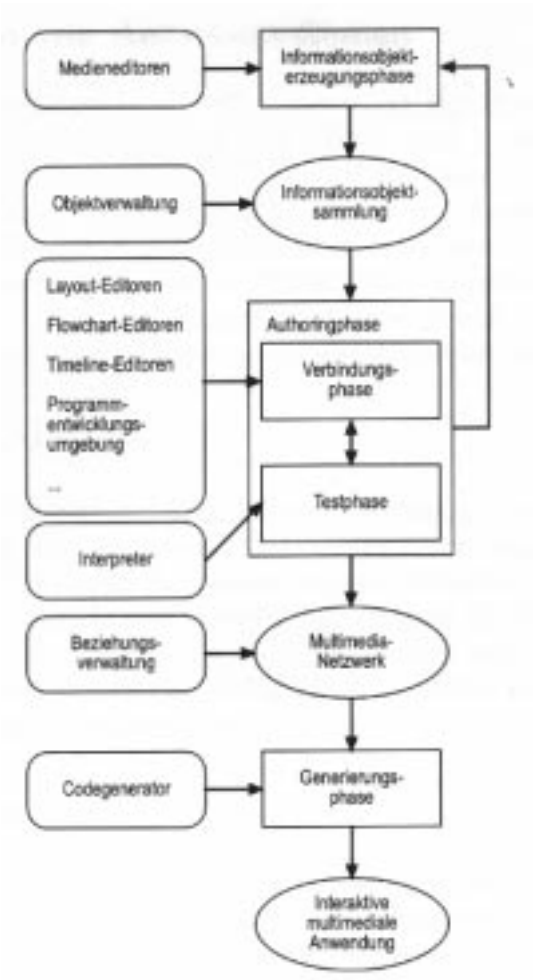


Abbildung 3.7: Anwendungsbereiche der Autorensysteme

Zunächst werden die einzelnen Informationsobjekte erzeugt und dann in der Informationsobjektsammlung abgelegt. Als Hilfsmittel zur Erstellung der Informationsobjekte dienen beispielsweise Texteditoren, Graphikeditoren oder Animationtools. In der nachfolgenden Strukturierungsphase werden die

Beziehungen zwischen den einzelnen Objekten definiert. Beliebige Ausschnitte der Anwendung kann sich der Entwickler in der Testphase ansehen und dann entscheiden, ob er mit seinem Ergebnis zufrieden ist oder nicht. Diese beiden Phasen stellen die eigentliche Authoring-Phase dar. Ergebnis der Authoring-Phase ist ein Multimedia-Netzwerk, das von der Beziehungsverwaltung des Autorensystems verwaltet wird. In der Generierungsphase schließlich wird der editierbare Programmcode erzeugt. Am Ende der Entwicklung steht dann das ausführbare Programm. In sogenannten Autorensystemen werden Werkzeuge zusammengefaßt, die von der Layout- bis zur Programmierungsphase eingesetzt werden. Während in einigen Autorensystemen die Programmierungsphase in die Verbindungsphase integriert ist, ist bei anderen Systemen kein Codegenerator vorhanden. Einige Systeme bieten zusätzlich noch Informationsobjekterzeugungswerkzeuge. Mit Hilfe von Autorensystemen soll dem (auch in der Herstellung von Multimedia-Systemen unerfahrenen) Benutzer die Entwicklung multimedialer Anwendungen erleichtert werden. Das geschieht mit graphisch-interaktiven Hilfsmitteln nach dem „What you see is what you get“-Prinzip(WYSIWYG-Prinzip). Der Autor muß sich nicht mit Programmiertechniken beschäftigen, sondern kann sich voll und ganz auf die Aufbereitung der Informationen und die Gestaltung der Benutzerschnittstelle konzentrieren. Die Autorensysteme unterstützen den Entwickler allerdings nicht im gesamten Software-Entwicklungsprozeß. Sie dienen dem Autor lediglich in der Implementierungs- und der Testphase als Hilfsmittel. Auf dem Markt sind mittlerweile zahlreiche Autorensysteme erhältlich. Sie lassen sich in frame-basierte, timeline-basierte und Flowchart-basierte Autorensysteme unterteilen.

3.5.2 Frame-basierte Autorensysteme

Zu den frame-basierten Autorensystemen gehört beispielsweise das Programm „HyperCard“(vgl. Abbildung 3.8), das ausschließlich für den Apple Macintosh verfügbar ist und im folgenden als Beispiel für frame-basierte Autorensysteme dient. HyperCard ermöglicht die Anordnung der darzustellenden Objekte auf sogenannten Karten(*Frames*). Diese Karten repräsentieren einen Bildschirm, wie er im Verlauf der Anwendung zu einem bestimmten Zeitpunkt zu sehen ist. Die eigentlichen Informationsobjekte wie Bilder oder Buttons werden auf den Karten angebracht. Sie können mit bestimmten Attributen (z.B. Name, Position auf dem Bildschirm, Größe) belegt werden. Ein komplettes Anwendungsprogramm besteht aus zahlreichen solcher Karten, die sich der Autor nacheinander ansehen und normalerweise nach dem WYSIWYG-Prinzip gestalten kann. Ein weiterer Vertreter der frame-basierten Autorensysteme ist z.B. „ToolBook“ für den PC.

3.5.3 Timeline-basierte Autorensysteme

Das bekannteste time-line-orientierte Autorensystem ist „Macromedia Director“(vgl. Abbildung 3.9). Director bedient sich der Film-Metapher. Der Autor übernimmt bei der Multimedia-Produktion die Rolle des Regisseurs. Wie beim Film besteht das zu erstellende Softwareprodukt hier aus vielen Einzelbildern, die erst zusammengesetzt den kompletten Film ergeben. Die wichtigsten Bestandteile des „Director-Studios“ sind die Bühne(vgl. Abbildung 3.9, oben), das Steuerpult(vgl. Abbildung 3.9, oben rechts), die Besetzung und das Drehbuch(vgl. Abbildung 3.9, unten).

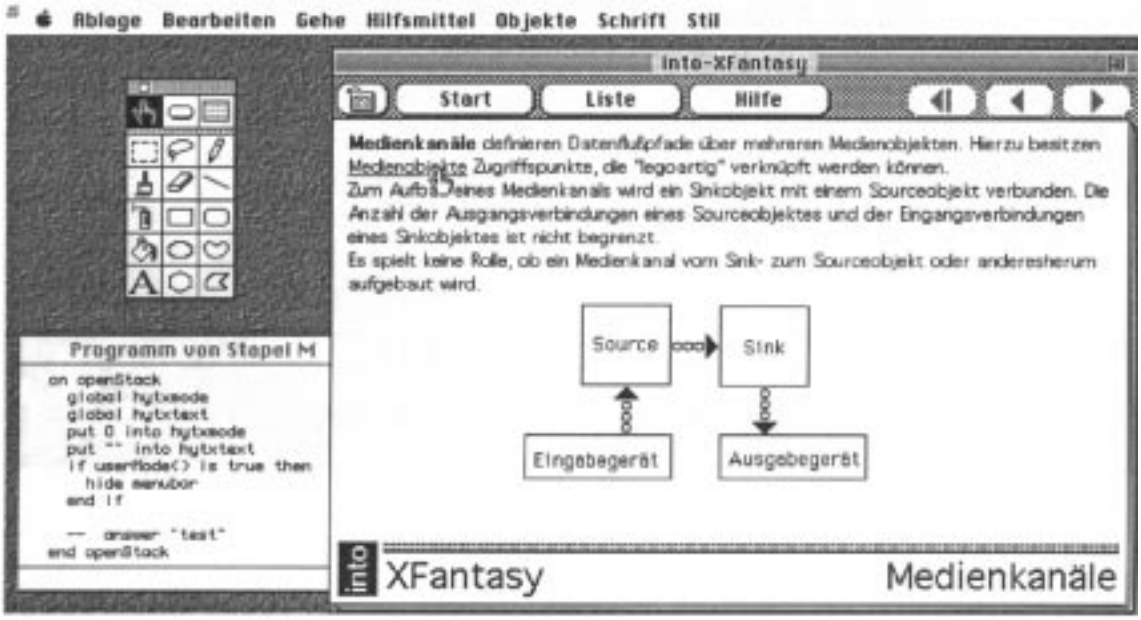

Ablage Bearbeiten Gehe Hilfsmittel Objekte Schrift Stil
into-XFantasy
Start
Liste
Hilfe
Medienkanäle definieren Datenflußpfade über mehreren Medienobjekten. Hierzu besitzen Medienobjekte Zugriffspunkte, die "legoartig" verknüpft werden können.
Zum Aufbau eines Medienkanals wird ein Sinkobjekt mit einem Sourceobjekt verbunden. Die Anzahl der Ausgangsverbindungen eines Sourceobjektes und der Eingangsverbindungen eines Sinkobjektes ist nicht begrenzt.
Es spielt keine Rolle, ob ein Medienkanal vom Sink- zum Sourceobjekt oder anderesherum aufgebaut wird.
Source
Sink
Eingabegerät
Ausgabegerät
Programm von Stapel M
on openStack
global hytxmode
global hytxtext
put 0 into hytxmode
put "" into hytxtext
if userMode() is true then
hide menubar
end if
-- answer "test"
end openStack
into
XFantasy
Medienkanäle

Abbildung 3.8: Benutzungsoberfläche von HyperCard

Im Fenster, daß die Bühne repräsentiert, läuft der Film ab. Mit dem Steuerpult kann der Ablauf des Films beeinflußt werden. In der Besetzung sind die einzelnen Darsteller(Informationsobjekte) abgelegt. Im Drehbuch sind die einzelnen Darsteller linear auf einer Zeitsachse angeordnet. Die Darsteller zusammengefaßt ergeben die Einzelbilder des Films, die wiederum nacheinander abgespielt den kompletten Film darstellen. Die Veränderung der Position der einzelnen Objekte auf dem Bildschirm erfolgt wie auch bei den frame-basierten Autorensystemen direkt oder über Menüs.

3.5.4 Flowchart-basierte Autorensysteme

In Flowchart-basierten Autorensystemen werden Informationsobjekte durch Ikonen repräsentiert. Kanten zwischen zwei Objekten stellen dabei den zeitlichen Ablauf dar. Ist ein Objekt Ausgangspunkt mehrerer Kanten, entscheidet eine Benutzerinteraktion, welches der Objekte als nächstes aufgerufen wird. Funktional zusammenhängende Objekte können zu einem Objekt zusammengefaßt und in ein externes Diagramm ausgelagert werden, was die Übersichtlichkeit verbessert. Auch bei den Flowchart-basierten Autorensystemen kann die Anordnung der Objekte auf dem Bildschirm graphisch-interaktiv erfolgen. Das bekannteste Flowchart-basierte Autorensystem ist Authorware Professional(vgl. Abbildung 3.10).

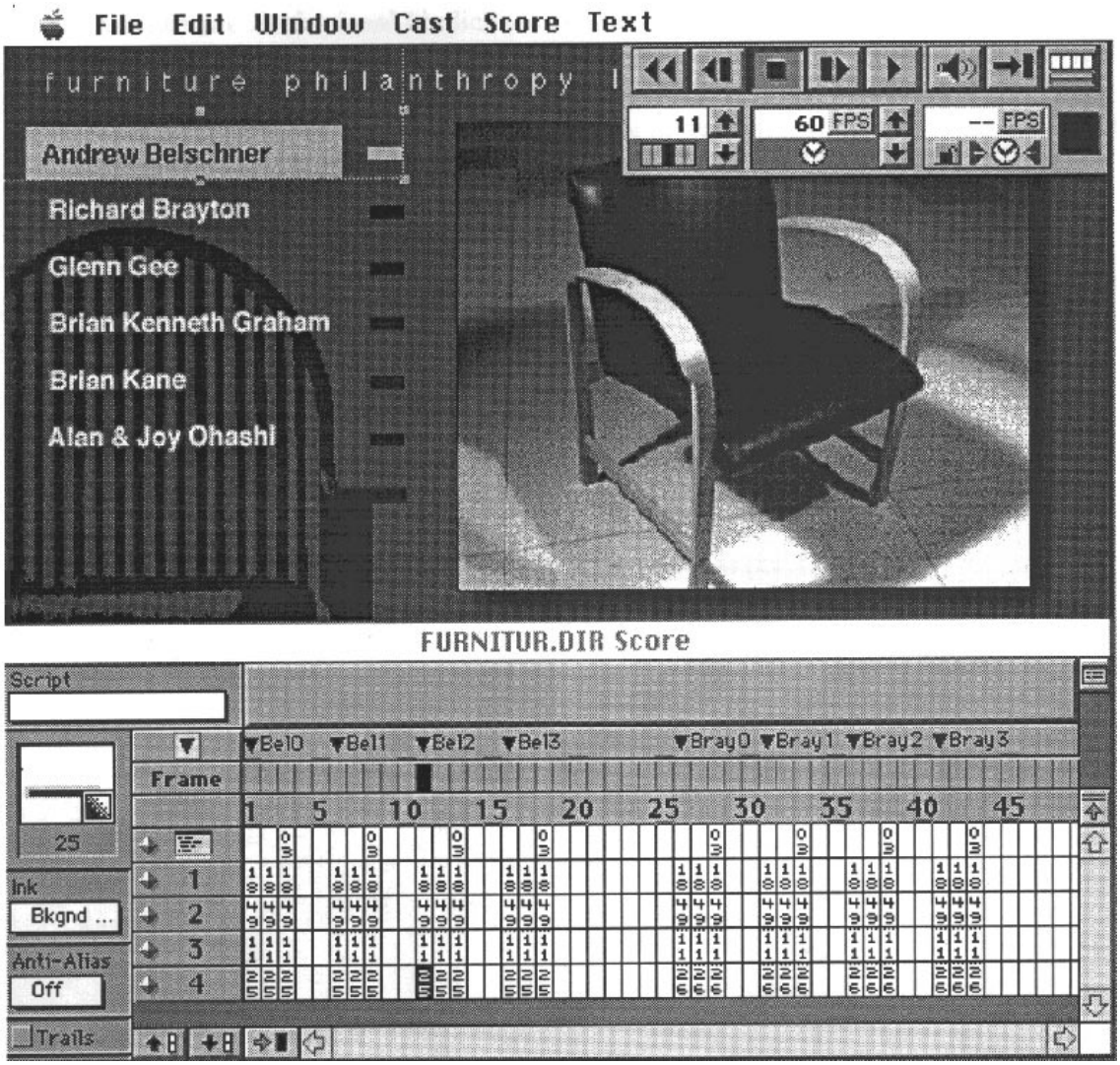


Abbildung 3.9: Benutzeroberfläche von Macromedia Director

3.5.5 Bewertung

Da in den letzten Jahren unzählige Autorensysteme auf den Markt gekommen sind, ist man bemüht, diese miteinander zu vergleichen. Dabei kommt man zu dem Ergebnis, daß die Autorensysteme der drei Kategorien je nach dem zu erstellendem Multimedia-Produkt Vor- und Nachteile haben. Demnach sind framebasierte Autorensysteme besonders gut für die Erstellung von eher statischen Hypermedia-Dokumenten geeignet. Für die Erstellung dynamischer interaktiver Multimedia-Softwareprodukte sollten hingegen timeline-basierte Autorensysteme verwendet werden. Bei der Einbindung von nicht-zeitlichen Beziehungen in eine graphische Repräsentation und bei der Handhabung von Informationsobjekten mit unbestimmter Lebenszeit, wie z.B. Live-Videos weisen diese jedoch ebenfalls Schwächen auf. Mit Flowchartbasierten Autorensystemen können diese Probleme besser behandelt werden. Der Nachteil dieses Ansatzes liegt nun wiederum darin, daß komplexe zeitliche Beziehungen im allgemeinen nicht darstellbar sind. Einige Autorensysteme stellen in der Verbindungsphase eine spezielle Programmiersprache zur

Verfügung. Das widerspricht allerdings dem eigentlichem Ziel der Autorensysteme, auch Personen ohne Programmierkenntnisse die Erstellung multimedialer Anwendungen zu ermöglichen. Ein weiteres Problem der Autorensysteme ist die meistens ungenügende Unterstützung bei der Integration von Interaktionsformen, sowie die fehlende Erweiterbarkeit von neuen Ein- und Ausgabetechnologien. Weitere Informationen zu Multimedia-Autorensystemen finden sich in [Bol95] und [Bol98].

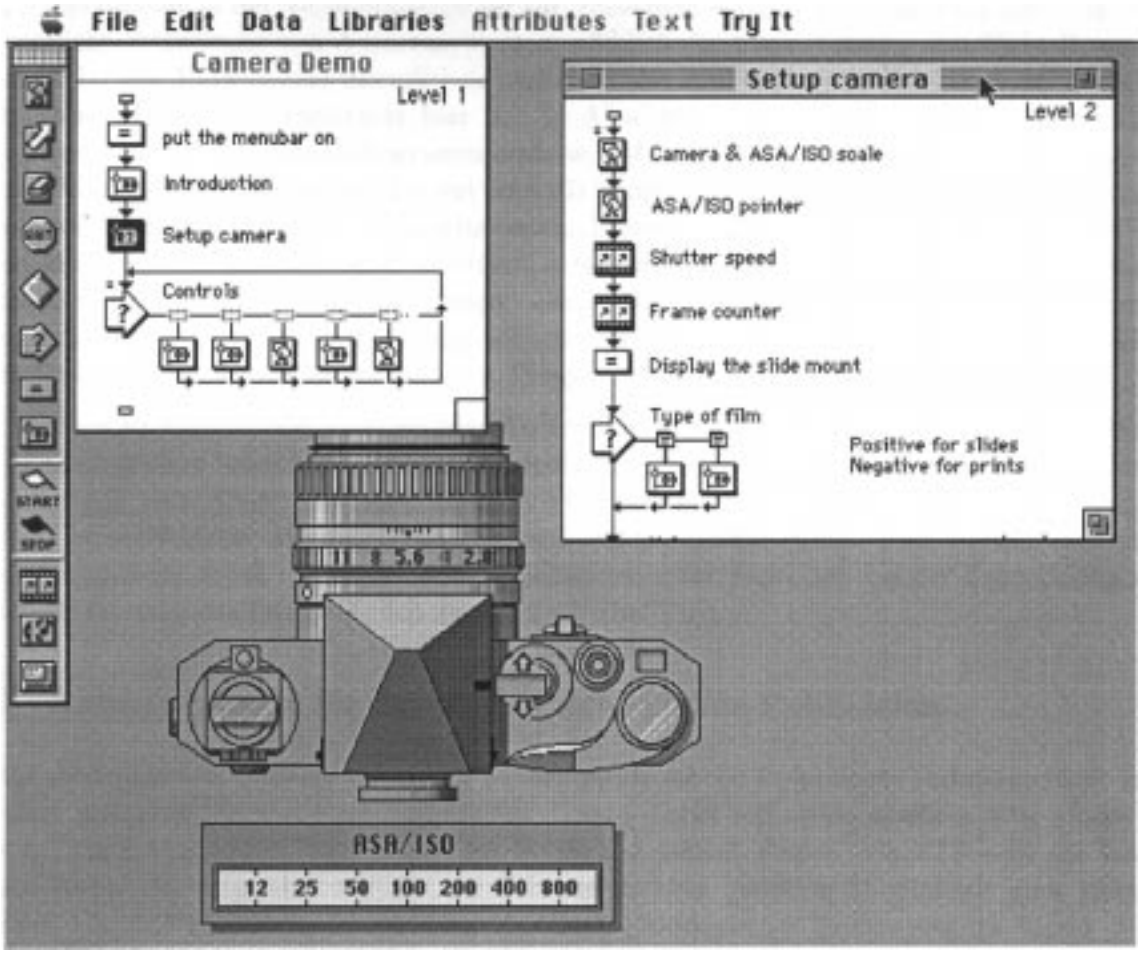


Abbildung 3.10: Benutzeroberfläche von Authorware Professional

3.6 Abschließende Anmerkungen

Der Bezug dieses Artikels zur Projektgruppe „Virtuelle naturwissenschaftlich-technische Labore im Internet“ ist folgendermaßen zu begründen. Da in der Projektgruppe ein Multimedia-Produkt erstellt werden soll, ist eine Auseinandersetzung mit „Multimedia-Software-Engineering“ und deren speziellen Charakteristiken absolut notwendig. Um diese Charakteristiken zu erkennen, muß zunächst das klassische Software-Engineering behandelt werden. Die abschließende Einführung in Multimedia-Autorensysteme ist insofern hilfreich, als daß in der Projektgruppe praktisch mit dem Autorensystem „Director“ gearbeitet werden wird.

Kapitel 4

Projektmanagement und Organisation

4.1 Zusammenfassung

Autor: Mario Sachteleben

In der Software-Entwicklung geht es darum, Software-Produkte in der erforderlichen Güte zu erstellen. Um dieses Ziel zu erreichen, sind für die Unternehmensführung einige Aspekte von großer Bedeutung. Die Planung ist für das Management ein sehr wichtiger Aufgabenbereich, dort wird definiert, was , wie, womit, wann zu machen ist. Dies wird im Voraus festgelegt, sollte sich aber gegebenenfalls dynamisch anpassen. Diese Tätigkeit wird vom Projektleiter in Kombination mit der Unternehmensführung durchgeführt. Im Abschnitt Organisation wird verdeutlicht, welche Möglichkeiten es gibt, ein Software-Unternehmen zu strukturieren und welche Vor -und Nachteile die einzelnen Ansätze bieten. Im Aufgabenbereich Personalwesen werden einige Kriterien genannt, mit dessen Hilfe die Zusammenarbeit zwischen den an der Software-Entwicklung beteiligten Personen optimiert werden kann. Diese Aspekte bekommen häufig nicht genügend Aufmerksamkeit. Diese Mißachtung ist des öfteren eine Erklärung für das Scheitern einiger Projekte. Diese Aspekte spielen auch bei der Leitung eine übergeordnete Rolle, da sehr hochqualifizierte spezialisierte Mitarbeiter geführt werden müssen. Die uneingeschränkte Kommunikation bekommt hier eine zentrale Bedeutung zugesprochen. Und schließlich wird während der Kontrolle ein Soll-Ist Vergleich durchgeführt, um den Stand der Entwicklung überblicken zu können.

4.2 Einleitung

Wesentliche Ziele des Software-Projektmanagements bestehen darin, die Kosten, welche bei der Erstellung der Software anfallen, zu minimieren und außerdem die vom Kunden geforderte oder für den Markt erforderliche Qualität der Software zu sichern. Indirekt beinhaltet das natürlich auch das Ziel, die Produktivität zu erhöhen. Die Güte des Software-Produktes ist auch von der Qualität des Managements abhängig. Die Führungskräfte haben dafür zu sorgen, daß diese Ziele durch die Mitarbeiter erreicht werden. Dazu werden fünf Funktionen teilweise sequentiell durchlaufen, es gibt aber auch gewisse Verzahnungen zwischen den einzelnen Funktionen: Planung, Organisation, Personalauswahl, Leitung und schließlich Kontrolle. Auf diese Aufgabenbereiche wird im folgenden näher eingegangen.

4.3 Planung

Planung ist ein systematisch-methodischer Prozeß der Erkenntnis und Lösung von Zukunftsproblemen nach [Sch95].

Es wird vor Beginn eines Projekts festgelegt, auf welchem Wege, mit welchen Schritten, in welcher zeitlichen und sachlogischen Abfolge, unter welchen Rahmenbedingungen und mit welchen Kosten und Terminen ein Ziel erreicht werden soll. Die Planung muß sich allerdings dynamisch anpassen, wenn sich die Entwicklung oder die Rahmenbedingungen ändern.

Bei der Planung von Software-Entwicklungen sind drei Abstraktionsebenen zu unterscheiden:

- Es muß überlegt werden, wie der Ablauf der Software-Entwicklung beschrieben werden soll, welche Standard-Prozeßelemente es gibt und wie ihr Zusammenwirken beschrieben werden soll. Solche Überlegungen führen zu einer **Prozeß-Architektur**.
- Es muß das generelle Vorgehen bei der Entwicklung festgelegt werden. Ergebnis hiervon ist das **Prozeß-Modell** oder Vorgehensmodell.
- Für jedes zu entwickelnde Softwareprodukt wird ein Projektplan aufgestellt. Verantwortlich für die Korrektheit dieses Dokuments ist der Projektleiter. Der Projektplan ist vom Prozeß-Modell abhängig.

4.3.1 Aufbau von Projektplänen

Ein Projektplan ist eine Erweiterung des ausgewählten Prozeß-Modells. Jeder Prozeß wird in zu erledigende Aufgaben unterteilt. Jede Aufgabe ist in sich abgeschloßen und kann innerhalb eines bestimmten zeitlichen Intervalls ausgeführt werden. Für jede Aufgabe sind festzulegen:

- Name der Aufgabe
- Erforderliche Zeit zur Erledigung dieser Aufgabe
- Zuordnung von Personal und Betriebsmitteln, die die Arbeit durchführen
- Kosten und Einnahmen, die mit dem Vorgang (der Aufgabe) zusammenhängen

Es muß zunächst eine Aufwandsabschätzung für das Gesamtprojekt durchgeführt werden, das Ergebnis wird auf jeden Vorgang verteilt. Anschließend gilt es zu planen, mit welchen Betriebsmitteln und welchem Personal die Aufgabe erfüllt werden soll.

Ein Beispiel eines Projektplans ist in Abbildung 4.1 zu sehen.

Des weiteren werden noch Meilensteine zur Projektüberwachung und Motivation des Projektteams festgelegt. Sie kennzeichnen den Beginn und Abschluß eines Vorgangs oder einer Phase (mehrere Vorgänge). Damit der Entwicklungsfortschritt wirksam kontrolliert werden kann, müssen Meilensteine folgende Anforderungen erfüllen:

1.Einleitung 1.1Zweck des Projektplans 1.2 Projektüberblick und Motivation 1.3 Entwicklungsphilosophie
2. Grundlagen 2.1 Vertragliche Anforderungen an Projektdurchführung 2.2 Vertragliche Anforderung an Lösung 2.3 Standart, Randbedingungen
3. Beschreibung des Projekts 3.1 Arbeitsumfang 3.2 Annahmen 3.3 Lieferumfang 3.4 Abnahmeprozedur 3.5 Resultate, die nicht zum Lieferumfang gehören
4. Entwicklungsplan 4.1 Aufteilung in Arbeitspakete 4.2 Netzplan mit Aktivitäten und Terminen 4.3 Budget 4.4 Kritische Punkte/Risiken
5. Anforderungen an die Umgebung 5.1 Rechnersysteme, Software 5.2 Leistungen des Auftraggebers 5.3 Leistungen externer Lieferanten
6. Entwicklungsprozess 6.1 Phasen der Entwicklung 6.2 Dokumentation 6.3 Qualitätskontrolle
7. Projektorganisation 7.1 Schnittstelle zum Auftraggeber 7.2 Schnittstelle zur Firmenorganisation 7.3 Schlüsselpersonen 7.4 Organisation während der Projektphasen
8. Standards für die Entwicklung 8.1 Konfigurationskontrolle 8.2 Programmierrichtlinien 8.3 Einsatz von Werkzeugen 8.4 Projektspezifische Abweichungen von Firmen-Standards

Abbildung 4.1: Ein Beispiel für ein Projektplan

- **Überprüfbarkeit**

  Mit Erreichen eines Meilensteins ist ein Teilprodukt auch wirklich fertiggestellt.

- **Kurzfristigkeit**

  Aufgaben, die mit einem Meilenstein abschließen, müssen in einem relativ kurzen Zeitintervall, etwa ein bis vier Wochen, durchführbar sein.

- **Gleichverteilung**

  Meilensteine müssen kontinuierlich über die Dauer eines Projektes verteilt werden, damit eine Kontrolle des Entwicklungsstandes eines Projektes möglich wird.

Zwischen Vorgängen, aber auch zwischen Meilensteinen, bestehen fachliche, terminliche und personelle Abhängigkeiten. Um diese sichtbar zu machen, ordnet man die Vorgänge und Meilensteine grafisch in einen Netzplan an. Eine Aufstellung von Netzplanarten ist in Abbildung 4.2 zu sehen.

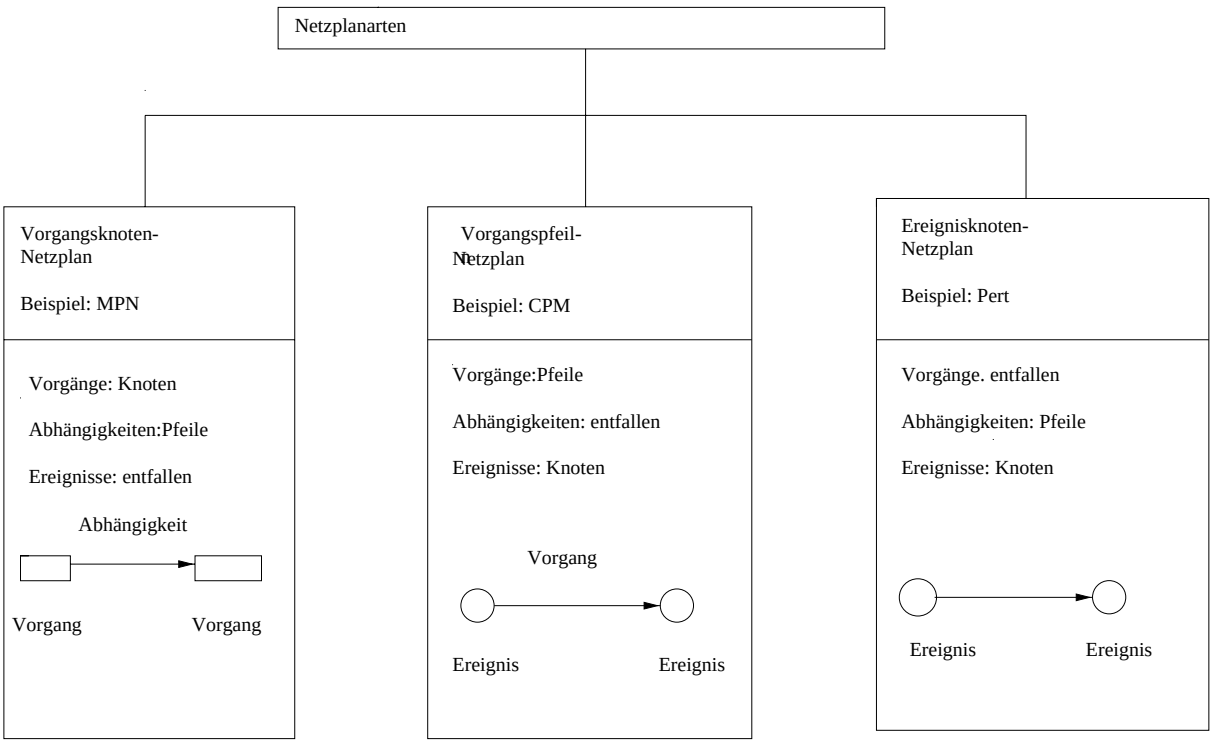


Abbildung 4.2: Netzplanarten im Vergleich

Bei den CPM-Netzplänen werden die Ereignisse, das sind Meilensteine durch einen konkreten Vorgang in ein Nachfolgerereignis überführt. Die Abhängigkeiten zwischen den einzelnen Vorgängen entfallen.

Bei den Pert-Netzplänen entfallen Vorgänge und die Meilensteine werden in Abhängigkeit voneinander dargestellt.

4.3.2 Zeitplanung

Hier wird die Zeitplanung mit MPM (Metra Potential Method) Netzplänen näher betrachten, weil sie sehr häufig Verwendung finden. Der prinzipielle Aufbau dieser Pläne wird durch Abbildung 4.3 illustriert.

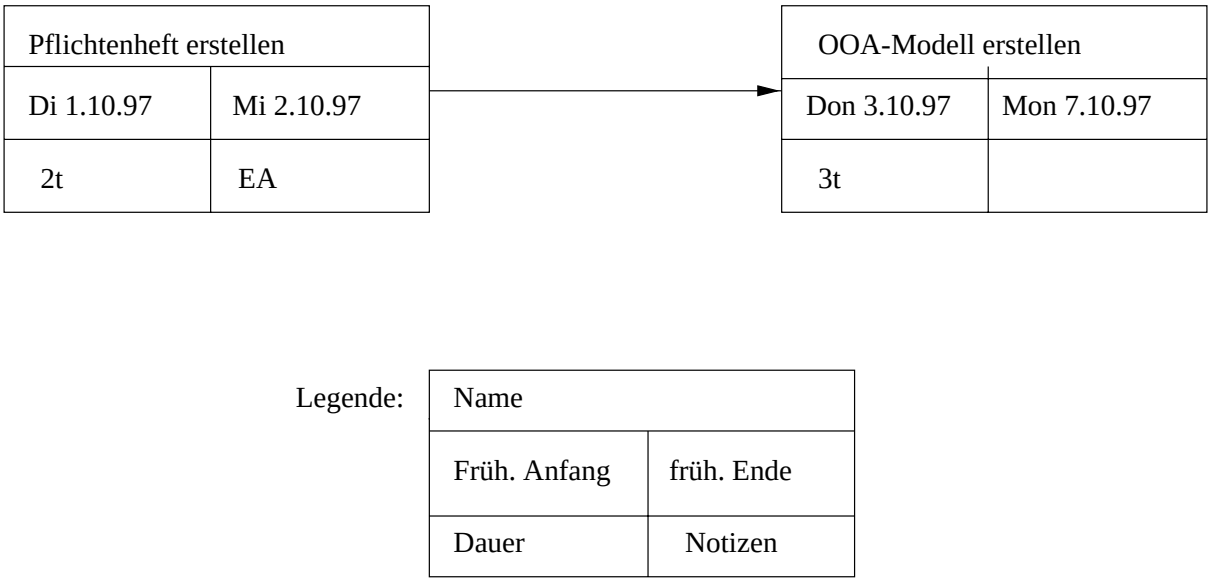


Abbildung 4.3: Netzplandarstellung

Ein Meilenstein wird durch einen Doppelkasten(Kasten um den Kasten) oder mittels Fettdruck dargestellt. Es wird zwischen vier Vorgangsbeziehungen unterschieden:

- Normalfolge: Ende-Anfang(EA)
- Anfangsfolge: Anfang-Anfang(AA)
- Endfolge: Ende-Ende(EE)
- Sprungfolge: Anfang-Ende(AE)

Die Pufferzeit ist die Differenz zwischen dem frühesten und spätesten Termin, an dem ein Vorgang beginnen kann. Nur eine zeitliche Verzögerung innerhalb dieses Zeitintervalls ist möglich, ohne das Gesamtprojekt zu gefährden.

Ein Vorgang, der keine Pufferzeit besitzt, wird als kritischer Vorgang bezeichnet. Stehen mehrere kritische Vorgänge in direkter Beziehung zueinander, so nennt man dies im Netzplan den kritischen Pfad. Er wird farbig oder mittels Fettdruck dargestellt. Diese sollten besonders sorgfälltig überwacht werden, damit das Projekt nicht gefährdet wird und termingerecht fertig gestellt werden kann.

4.3.3 Projektplanung

Zunächst wird ein Prozeß-Modell ausgewählt, an dem sich die Projektführung orientieren soll. Daraus ist dann der Projektplan abzuleiten. Da für jede zu verrichtende Aufgabe ein Zeitintervall zur Erledigung

angegeben werden muß, ist eine Aufwandsabschätzung für das Gesamtprojekt durchzuführen. Zusätzlich sollten empirische Daten vergangener Projekte über Aufwand und Verteilung des Aufwands auf die einzelnen Phasen vorliegen. Ausgehend vom Aufwand pro Vorgang gilt es zu planen, mit welcher Personalkapazität der Aufwand bewältigt werden soll. Ausgehend von diesen Überlegungen werden die Zeitintervalle zur Erledigung der Einzelaufgaben für das Projekt festgelegt.

Anschließend erfolgt eine Netzplandurchrechnung. Es werden die kritischen Pfade ermittelt und Überlegungen über mögliche zeitliche Optimierung getätigt. Um eine Verzögerung des Gesamtprojektes zu verhindern, sollten Pufferzeiten für risikoreiche Aufgaben eingeplant werden. Risikoreiche Aufgaben sind solche, wo häufig unerwartete Zwischenfälle auftreten, die zu einer zeitlichen Verzögerung des Projektes führen.

Da nun die Terminplanung vorliegt, müssen die für jeden Vorgang notwendigen Ressourcen (Personal und Betriebsmittel) geschätzt und zugeordnet werden. Nun muß die Auslastung der Ressourcen überprüft und gegebenenfalls eine Bedarfsoptimierung durchgeführt werden. Um eine Kostenplanung durchführen zu können, müssen die Kosten den Ressourcen zugeordnet werden.

4.4 Organisation

In der Software-Entwicklung sollten nach Abschluß eines Projekts korrekte Programme entstanden sein. Aus terminlichen Gründen und wegen der hohen hohen Anforderungen an die zu erstellenden Programme, werden die Aufgaben in der Regel von mehreren Mitarbeitern vollzogen. Daher sind grundlegende Aufgaben zu erledigen:

- Der Arbeitsablauf zur Erstellung des Software-Produkts ist festzulegen, d.h die Ablauforganisation ist zu definieren. Aus dem gesamten Arbeitsablauf ergeben sich dann die zu erledigenden Einzelaufgaben und ihre Beziehungen zueinander.
- Diese Einzelaufgaben müssen dann auch koordiniert werden. Dazu wird eine geeignete Aufbauorganisation festgelegt.

4.4.1 Betriebsstruktur

Die Organisation wird in fünf Teile aufgeteilt,was in Abbildung 4.4 zu sehen ist.

Die Mitarbeiter, die direkt an der Erstellung der Produkte beteiligt sind, bilden den betrieblichen Kern (z.B Programmierer).

Die strategische Spitze trägt Verantwortung dafür, daß die Unternehmung ihre Aufträge effektiv erfüllt. Sie hat drei Aufgabenbereiche. Durch persönliche Weisung werden Ressourcen verteilt, Aufträge vergeben, Entscheidungen genehmigt, Konflikte gelöst, Strukturpläne erstellt, Personal eingestellt, Arbeitsleistung kontrolliert sowie Mitarbeiter motiviert und belohnt. Diesen Aufgabenbereich übernimmt sowohl die strategische Spize, als auch die Mittellinie. Der zweite Aufgabenbereich umfaßt die Vertretung der Organisation nach außen, d.h die Aufrechterhaltung ihrer Beziehungen zur Umwelt. Die strategische Planung der Organisation ist der dritte Aufgabenbereich.

Die strategische Spitze ist über eine Folge von Führungskräften (der Mittellinie) mit dem betrieblichen Kern verbunden. Diese Ketten verlaufen linear von oben nach unten. Die Führungskräfte der

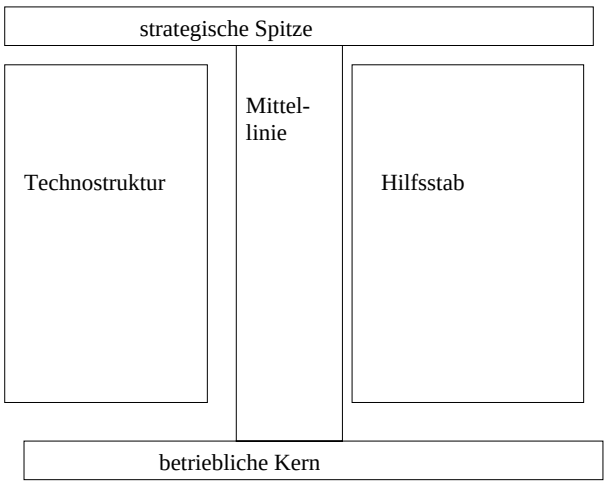


Abbildung 4.4: Die fünf Teile einer Organisation

Mittellinie verrichten eine Reihe von Tätigkeiten im Rahmen des ersten Aufgabenbereichs oberhalb und unterhalb der eigenen Position.

Abwärts fließen Entscheidungen über die Ressourcen für die eigenen Organisationseinheiten, über die auszuarbeitenden Vorschriften und Pläne sowie über Arbeiten, für deren Durchführung die Einheiten zuständig sind. Von unten nach oben fließen häufig Rückmeldungen, Änderungsvorschläge oder Entscheidungen, für die eine Genehmigung notwendig ist. Des weiteren existiert eine Gruppe von Personen, die wir als Technostruktur bezeichnen. Diese Gruppe setzt sich aus Analytikern zusammen. Sie gestalten, planen und verändern den betrieblichen Arbeitsablauf. Sie können allerdings auch die betroffenen Mitarbeiter ausbilden. Der Hilfsstab unterstützt die Unternehmung außerhalb des betrieblichen Ablaufs, dazu zählen z.B Telefonzentrale, Kantine, Rechtsabteilung etc.

4.4.2 Gestaltung der Aufbauorganisation

Zur Organisationsgestaltung bietet sich ein kombiniertes top-down und bottom-up Verfahren an:

- Sämtliche Aufgaben werden zusammengestellt, die unter Beachtung der übergeordneten organisatorischen Voraussetzungen anfallen.
- Je nach Grad der Spezialisierung werden sie den unterschiedlichen Positionen zugeordnet. Es wird festgelegt, inwieweit diese formalisiert werden soll und welche Ausbildung erforderlich ist.
- Es wird festgelegt, wie viele Stellen in den Einheiten auf der untersten Ebene zusammengefaßt werden sollen. Dann wird entschieden, welche und wieviele Einheiten jeweils in den nächst höheren Ebenen gruppiert werden, bis die vollständige Hierarchie (Aufbauorganisation) vorliegt.
- Die Aufbauorganisation wird ausgebaut und es werden Entscheidungsbefugnisse zugeordnet.

Durch Gruppierung von Positionen und Einheiten werden Arbeitsbereiche in einer Organisation koordiniert. Gruppierungen haben folgende Auswirkungen:

- Für alle Gruppenmitglieder wird ein gemeinsames Weisungssystem geschaffen. Jede Einheit wird durch eine Führungskraft geleitet, die für die Tätigkeiten innerhalb der Einheit verantwortlich ist.
- Innerhalb einer Gruppe werden Ressourcen gemeinsam benutzt. Das fördert die informellen Kontakte, und einzelne Aufgaben in der Gruppe werden koordiniert.
- Innerhalb einer Gruppe entwickeln sich gemeinsame Leistungsmaßstäbe. Dies fördert die Bereitschaft der Mitglieder, ihre Aktivitäten zu koordinieren. Außerdem bildet dies die Grundlage für die Standardisierung der Arbeitsprodukte.

Zwischen verschiedenen Einheiten kann es allerdings zu Koordinationsproblemen kommen. Es gibt verschiedene Möglichkeiten zu gruppieren z.B nach Zeit, Qualifikation, Arbeitsprozeß, etc.

Je besser die Mitarbeiter ausgebildet sind, umso weniger müssen sie kontrolliert oder ihnen Hilfestellung gegeben werden und desto größer können ihre Arbeitseinheiten groß sein. Eine Arbeitseinheit muß umso kleiner sein, je stärker die gegenseitige Absprache zur Koordination von voneinander abhängigen Aufgaben ist. Zu große Gruppen fördern häufig die Cliquenbildung, wobei hier der Kommunikationsaufwand nicht unterschätzt werden darf. Des weiteren sollte die optimale Anzahl von Mitarbeitern rechtzeitig bekannt und verfügbar sein, denn eine verspätete Aufstockung führt fast immer zu einer Terminverschiebung. Große Einheiten sollten gebildet werden, wenn:

- die Aufgaben in einer Einheit ähnlich geartet sind,
- die Mitarbeiter selbständig sein und sich verwirklichen wollen,
- die Informationsvermittlung von unten nach oben optimal funktionieren soll.

Kleinere Einheiten sollten gebildet werden, wenn komplexe, interdependente Aufgaben eine gegenseitige Abstimmung erfordern.

4.4.3 Projektstrukturen

Es wird im [Bal98]zwischen fünf Organisationsstrukturen unterschieden:

- Einfachstruktur
- Maschinenbürokratie
- Spartenstruktur
- Projektstruktur
- Profibürokratie

Für das Software-Management sind besonders die Projektstruktur und die Profibürokratie relevant, daher werde bekommen diese im folgenden mehr Aufmerksamkeit zugesprochen.

In der **Projektstruktur** wird mit Projektteams entwickelt, in diesen Teams sind Mitarbeiter verschiedener Disziplinen tätig. Es findet eine hohe horizontale Aufgabenspezialisierung statt. Auf der einen

Seite gibt es eine Tendenz, die Mitarbeiter in funktionalen Einheiten einzusetzen, auf der anderen Seite werden Mitarbeiter in marktorientierten Projektteams zusammengefaßt. Des weiteren kommen Kontaktinstrumente zur Förderung der gegenseitigen Abstimmung zum Einsatz. Projektteams werden von Projektmanagern geleitet. Diese Manager sind meistens keine klassischen Führungkräfte, sie kümmern sich vielmehr um Kontaktpflege und Verhandlungsführung. Viele dieser Manager könnten aufgrund ihrer Qualifikationen und Fähigkeiten im Team mitarbeiten. Die Entscheidungsbefugnisse sind auf Führungskräfte und Nichtführungskräfte aller Hierarchieebenen verteilt. Diese Organisatiosstruktur kann sich dynamisch der Umwelt anpassen. Sie lebt von Projekt zu Projekt, d.h nach Beendigung eines Projektes werden die Teams aufgelöst. In einem Team ist es erforderlich, daß jedes Mitglied seine eigenen Interessen denen des Teams unterordnet. Ein weiterer Nachteil ist die Ineffizienz, die durch den hohen Kommunikationsanteil und ungleichmäßige Arbeitsauslastung entsteht. Diese Projektstruktur ist für einmalige Projekte mit anspruchsvollen Innovationen gut geeignet.

Bei der **Profibürokratie** werden im betrieblichen Kern komplexe Arbeiten durchgeführt, um Standardprodukte zu erstellen oder Standarddienstleistungen anzubieten. Sie läßt sich wie folgt charakterisieren:

- Die Koordination erfolgt durch Standardisierung der Qualifikationen.
- Der betriebliche Kern ist der wichtigste Organisationsteil. Die Ausführung der Arbeiten erfolgt durch professionelle Mitarbeiter, die entsprechend ausgebildet sind.
- Jeder Mitarbeiter erledigt seine Arbeit weitgehend alleine und in Eigenverantwortung.
- Die Arbeitsaufgaben sind horizontal spezialisiert, aber vertikal erweitert.
- Die Profibürokratie besteht in einer komplexen, aber stabilen Umwelt. Die technischen Systeme sind nicht regulativ und nicht kompliziert.
- Die Struktur ist funktions -und marktorientiert.

Die Organisationsstruktur ist im wesentlichen bürokratisch, wobei eine Koordination über bestimmte Ausführungsstandards erreicht wird. Diese Struktur ist immer dann sinnvoll, wenn:

- Der betriebliche Kern aus hochqualifizierten Mitarbeitern besteht, welche die schwierigen, aber gut definierten Verfahren anwenden.
- Die Umwelt so komplex ist, daß solche Verfahren Anwendung finden müssen.
- Die Umwelt so stabil ist, daß die Qualifikation der Mitarbeiter gut zu definieren und zu standardisieren ist.

Die Profibürokratie ist gut geeignet für die Erstellung von Standardprodukte, aber weniger für die Erstellung neuer Produkte.

4.5 Personal

Untersuchungen haben ergeben, daß ein Großteil gescheiterter Projekte nicht aufgrund technischer, sondern aus persönlichen zwischenmenschlichen Gründen gescheitert sind. Eine qualitativ hochwertige Software läßt sich häufig auf gute menschliche Zusammenarbeit zurückführen.

In der Softwaretechnik werden viele Spezialisten eingesetzt wie z.B Software-Entwerfer (abstraktes und konzeptionelles Denken), Implementierer (mathematisches Denkvermögen, Abstraktionsvermögen, Präzision), Software-Ergonomen (hohe Kommunikationsbereitschaft, Interdisziplinäres Wissen, didaktische Kenntnisse) etc. Hier ist nun die Einsatzfähigkeit eines jeden Mitarbeiters aufgrund der hohen Spezialisierung stark eingeschränkt. Aufgabe des Managements ist es, zum einen dafür zu sorgen, daß immer in ausreichendem Maße Mitarbeiter mit der notwendigen Qualifikation verfügbar sind, und zum anderen müssen dafür Rahmenbedingungen geschaffen werden, daß diese Spezialisten miteinander kommunizieren.

4.5.1 Allgemeine Qualifikationen

In der Software-Technik sollte bei der Einstellung neuer Mitarbeiter auf einige allgemeine Qualifikationen geachtet werden:

- Fähigkeit zum Abstrahieren
- Sprachliche und schriftliche Kommunikationsfähigkeit
- Teamfähigkeit
- Wille zum lebenslangen Lernen
- Kreativität
- Hohe Belastbarkeit
- Gute Englischkenntnisse
- Schreibmaschinenkenntnisse

4.5.2 Aufgaben und Aktivitäten

4.5.2.1 Stellen besetzen

Ein Software-Manager muß sich bei der Auswahl seiner potentiellen Mitarbeiter von deren Qualifikationen und deren Erfahrungen überzeugen. Es sollten jedoch noch weitere Faktoren beachtet werden:

- Motivation
- Engagement
- Selbständigkeit

- Gruppenaffinität
- Intelligenz
- Persönlichkeit

Schwächen auf einigen dieser Gebiete können durch Stärken auf anderen Gebieten wieder ausgeglichen werden. Qualifizierte Mitarbeiter können durch Neueinstellung, aber auch durch Versetzung innerhalb des eigenen Unternehmens gewonnen werden.

4.5.2.2 Integration neuer Mitarbeiter

Neue Mitarbeiter müssen mit den Entwicklungsrichtlinien-, methoden -und werkzeugen, aber auch mit den anderen Mitarbeitern vertraut gemacht werden. Der Manager eines Unternehmens ist in der Regel für die Integration neuer Mitarbeiter in das Unternehmen verantwortlich.

4.5.2.3 Weiterbildung von Mitarbeitern

Gerade in der Software-Technik ist die ständige Weiterbildung aufgrund der hohen Innovationsgeschwindigkeit, die zu einer Verdopplung des Wissens alle vier Jahre führt, unbedingt erforderlich.Nach [Bal98] benötigen Mitarbeiter in der Software-Technik pro Jahr zwei bis drei Wochen Vollzeit-Training, um den Stand der Technik auf dem jeweiligen Arbeitsgebiet zu halten. Werden neue Methoden und Werkzeuge eingeführt, dann ist dafür zusätzliche Zeit notwendig. Die zu vermittelnden Themen sollten unter Berücksichtigung der verwendeten spezifischen Werkzeuge, Techniken, Methoden und Rechnerhilfsmittel für die Entwicklung und das Management des Softwareprodukts festgelegt werden. Es kann auch erforderlich sein, die Schulung von Fertigkeiten und Vermittlung von Kenntnissen auf dem betreffenden speziellen Softwaregebiet einzuschließen. Zweckentsprechende Aufzeichnungen über die Schulung und Erfahrung sollten aufbewahrt werden.

4.5.2.4 Personalentwicklung

Ziel der Personalentwicklung ist es, für die Durchführung der aktuellen und der zukünftigen Entwicklungsaufgaben in ausreichender Qualität und Quantität Personal zu geringen Kosten zur Verfügung zu stellen. In der Software-Technik gibt es jedoch auf dem Gebiet der Personalentwicklung zwei Problembereiche, denen besondere Aufmerksamkeit zu widmen ist.

- Fluktuation von Mitarbeitern
- Qualifikationsprobleme bei nicht fortbildungsfähigen Mitarbeitern.

Für das erste Problem bieten sich folgende Lösungsvorschläge:

- Gute personenunabhägige Dokumentation
- Unabhägige Qualitätssicherung

- Attraktive Arbeitsbedingungen
- Realistische Erwartung bei der Personalplanung
- Mittelfrisitige Personalplanung

4.6 Leitung

In diesem Aufgabenbereich werden Tätigkeiten durchgeführt wie zum Beispiel Führung und Beaufsichtigung von Mitarbeitern, das Delegieren von Kompetenzen , das Koordinieren von Aktivitäten, das Fördern der Kommunikation, das Lösen von Konflikten und die Einführung von Innovation.

4.6.1 Teamarbeit

In der Software-Entwicklung besteht die Aufgabe des Managements darin, hochqualifizierte Mitarbeiter, die aufgrund von komplexen Aufgabenstellungen in Teams zusammengefaßt sind, zu führen. Innerhalb dieser Teams müssen die Mitarbeiter ständig miteinander kommunizieren, um ihre Aufgaben abzugleichen. Jedes Mitglied ist gleichrangig und hat gleiches Stimmrecht bei Abstimmungen. Verschiedene Mitglieder führen das Team von Zeit zu Zeit jeweils auf dem Gebiet, wo ihre Stärken liegen.

Es gibt Faktoren, die eingehalten werden sollten, um eine erfolgreiche Teamarbeit nicht zu gefährden: Ein Team braucht ein gemeinsames Ziel (das steigert die Motivation), Erfolgserlebnisse dürfen insbesondere in der Anfangsphase nicht ausbleiben(Zeichen setzen, daß das Team auf den vermeindlich richtigen Weg ist), Mitarbeiter brauchen das Gefühl unersetzbar zu sein (elite Team), außerdem braucht jedes Team die Herausforderung, ein besonders wertvolles Produkt schaffen zu dürfen (kein Wegwerfprodukt). Allerdings sollte auch darauf geachtet werden, daß kein unnötiger Funktionsumfang entwickelt wird. Ein Team kann nur erfolgreich arbeiten, wenn die Mitglieder von den persönlichen Eigenschaften wie z.B vom Charakter her zusammen passen. Nur dann sind sie fähig, sich als Team zu identifizieren, und können erfolgreich zusammenarbeiten. Deshalb sollte auf die Zusammenstellung der Teams besonders Einfluß genommen werden. Ein Team kann sich auch nur richtig entfalten, wenn es das nötige Vertrauen seitens der Unternehmensführung bekommt; zuviel Kontrolle wirkt einschränkend. Eine räumliche Trennung behindert das Wir-Gefühl. Es gilt natürlich auch der Spruch ‘Never change a winning team‘. Die Termine sollten so gesetzt werden, daß sie eingehalten werden können, die Teammitglieder aber auch nicht zu unausgelastet sind. Es sollte also die goldene Mitte getroffen werden. Ein gutes Team erkennt man an folgenden Eigenschaften:

- Niedrige Fluktationsrate
- Starke Identifikation eines jeden Mitglieds mit dem Team
- Freude an der Arbeit
- Bewußtsein einer Elitemannschaft

4.6.2 Kreativität

Gerade in der Software-Entwichklung ist ein hoher Grad an Kreativität erforderlich wie zum Beispiel beim Entwerfen einer geeigneten Systemarchitektur.

Orginelle Ideen sind dadurch charakterisiert, daß sie die prinzipiellen Denkansätze aus dem jeweiligen Problemfeld überspringen und neue vom Problemfeld weit entfernt liegende Ansätze aufnehmen und zu einer Lösung des jeweiligen Problems führen.

Die meisten Kreativitätstechniken werden innerhalb von Gruppen durchgeführt, da mehr Personen oft besser innovativ denken können. Grundsätzlich kann so verfahren werden, daß eine Idee (und mag sie auf den ersten Blick noch so ungeeignet aussehen) in den Raum geworfen wird. Die anderen Mitglieder in der Runde nehmen diese als Innovation auf und versuchen ihrerseits auf diesem Wege zu neuen Ideen zu gelangen. Um jeden Teilnehmer gleichrangig zu behandeln, kann auch von der mündlichen Diskussion abgegangen und das gleiche schriftlich durchgeführt werden.

4.7 Kontrolle

4.7.1 Aufgaben

Hier wird ein Soll-Ist Vergleich aufgestellt. Treten Abweichungen zum Plan auf, so werden geeignete Aktionen gestartet, um diese Abweichungen zu korrigieren. Der grundlegende Kontrollprozeß beinhaltet:

- Das Einrichten von Plänen und Standards.
- Das Messen der jeweiligen Version gegen diese Pläne und Standards.
- Die Korrektur der Abweichungen.

Die Kontrolle ist ein Rückkopplungssystem, das anzeigt, wieweit die Entwicklung bezogen auf das Produkt vorangeschritten ist. Der Kontrollprozeß muß in das Prozeßmodell und in die organisatorische Struktur integriert werden; es müssen die Zuständigkeiten in diesem Aufgabenbereich festgelegt werden.

Um eine Entwicklung zu kontrollieren, müssen vom Software-Management folgende Aufgaben durchgeführt werden:

- Standards entwickeln und festlegen
- Kontroll-und Berichtssystem etablieren
- Prozesse und Produkte vermessen
- Korrigierende Aktionen initiieren
- Loben und Tadeln

Für Entwicklungsberichte ist der Typ, die Häufigkeit, der Ersteller und der Empfänger festzulegen. Korrigierende Maßnahmen können sein: Das Ändern von Plänen, das Ändern des Produktumfangs oder andere Aktionen, solange bis die Einhaltung des Planes wieder möglich ist.

4.7.2 Konfigurationsverwaltung

Ein Software-Produkt besteht im allgemeinen aus einer Vielzahl von Software-Elementen. Um zu wissen, welches Software-Element zu welchem Software-Produkt gehört, beschreibt man die Zusammengehörigkeit durch eine Konfiguration. Eine Konfiguration ist eine Menge von Software-Elementen (mit jeweils gültigen Versionsangaben), die in einem bestimmten Zeitpunkt im Produktlebenszyklus in ihrer Wirkung und Schnittstellen aufeinander abgestimmt sind und gemeinsam eine geplante Aufgabe erfüllen sollen. Jedes Software-Element bekommt eine eindeutige Identifikation. Jede Änderung erzeugt ein neues Element mit einer neuen Identifikation. In einem Konfigurations-Identifikationsdokument (KID) wird aufgeführt, welche Software-Elemente (einschließlich Werkzeuge) zu einem Produkt gehören. Vorteil hiervon ist, daß man auf eine vorherige Konfiguration zurückgreifen kann, wenn eine aktuelle Konfiguration fehlerhaft ist.

Um dieselbe Möglichkeit bei der Entwicklung der Software zu haben, werden Zwischenergebnisse in Konfigurationen gespeichert, diese nennt man auch Referenzkonfiguration (baseline).

Eine Version kennzeichnet die Ausprägung eines bestimmten Software-Elements zu einem bestimmten Zeitpunkt. Die Versionsnummer besteht im allgemeinen aus zwei Teilen (nach [KF88]):

- der Release-Nummer
- und der Level-Nummer

Die Release-Nummer steht immer vor der Level-Nummer. Ein neues Software-Element enthält in der Regel die Versionsnummer 1.0. Bei jeder kleineren Änderung an dem Software-Element wird die Level-Nummer um 1 erhöht. Bei jeder gravierenden Änderung an dem Software-Element wird die Release-Nummer um 1 erhöht und zeitgleich die Level-Nummer auf Null gesetzt.

Das am häufigsten verwendete Modell zur Versionsverwaltung ist das Checkin/Checkout-Modell. Hier bekommt die bearbeitende Person eine Kopie des zu bearbeitenden Software-Elements, und es wird auch für diese Person während der Bearbeitungszeit reserviert. Es kann also keine anderer auf dieses schreibend zugreifen. Dies ist erst wieder möglich, wenn es mit Checkin wieder freigegeben wird. Eine Datei kann also zu einem bestimmten Zeitpunkt nur von einer Person verändert werden. Dadurch soll sie in konsistentem Zustand verbleiben.

4.8 Abschließende Anmerkungen

Nur wenn die fünf Hauptaufgaben korrekt hintereinander ausgeführt werden, kann ein Software-Produkt erfolgreich entwickelt werden. Nochmal zur Erinnerung, die Hauptaufgaben waren:

- Planung
- Organisation
- Personalauswahl
- Leitung

- und Kontrolle

Zum Schluß wird nochmal die Organisation und das Personal kurz angesprochen. Zunächst kommen wir zur Organisation. Es ist wichtig, daß wenn eine Hierarchie im Betrieb gebildet wird, man unbedingt darauf achten sollte, daß immer mehrere Pfade von unten nach oben sowie umgekehrt existieren. Wäre dem nicht so, dann könnten einzelne Personen eventuell den Datenfluß unterbrechen mit dem Ziel, die eigene Machtposition zu erhalten. Bei mehreren Pfaden durchläuft der Informationsfluß einen anderen Weg bis zum Ziel.

Beim Personal ist ein letzter Aspekt ganz wichtig, daß sich die Mitarbeiter, wenn es die Arbeit zuläßt (und natürlich nur dann!) auch mal über andere Dinge als das Fachliche reden. Das verbessert das Arbeitsumfeld und die Leute lernen sich besser kennen, was auch eine positive Wirkung auf die Produktivität hat.

Kapitel 5

Objektorientierte Softwareentwicklung mit der Unified Modeling Language (UML)

5.1 Zusammenfassung

Autor: Thomas Aden

Die Idee der Objektorientierung ist schon über 25 Jahre alt. Die Geschichte der objektorientierten Methoden der Softwareentwicklung begann jedoch erst ca. 1988. Nachdem sich einige Methoden etabliert hatten, die sich jedoch nicht unbedingt universell einsetzen ließen, begannen die bekanntesten Autoren der Methoden gemeinsam die Arbeit an der UML bei der Firma Rational. Diese ist eine universelle Notation für die Modellierung von Softwaresystemen, die sich vor allem auch auf alle Einsatzbereiche erweitern läßt. Ihre Elemente werden im letzten Abschnitt vorgestellt.

Für das Verständnis der vorgestellten Elemente, werden zunächst die Grundbegriffe der Objektorientierung erläutert. Hierzu zählen die Begriffe Klasse, Objekt, aber auch die Mechanismen, die der Objektorientierung eigen sind, wie z. B. Vererbung, Polymorphie, Assoziationen, Nachrichten bzw. Kommunikation, abstrakte Klassen und Behälterklassen.

Da die UML jedoch keine Beschreibung einer Entwicklungsmethode enthält, wird in Abschnitt 5.4 ein evolutionäres und iteratives Vorgehensmodell vorgestellt, das die Phasen Analyse, Design, Implementierung und operativer Einsatz innerhalb eines Zyklus beschreibt.

5.2 Einleitung

Diese Arbeit ist innerhalb der ersten Seminarphase des Projektes „Virtuelle naturwissenschaft-lich-technische Labore im Internet“ entstanden. Sie beschäftigt sich mit dem objektorientierten Softwareentwurf mit der Unified Modeling Language (UML), die zur Modellierung eines Software-Werkzeuges eingesetzt werden soll.

Der nachfolgende Teil beschreibt zunächst die Grundbegriffe der Objektorientierten Softwareentwicklung, ohne auf unterschiedliche Methoden einzugehen. Der darauffolgende Teil beschreibt dann den Prozeß der objektorientierten Softwareentwicklung. Dabei wird vertiefend auf die von Oestereich [Oes97] beschriebene Methode eingegangen, andere werden kurz vorgestellt. Der letzte Abschnitt zeigt anschließend die Modellelemente der UML anhand von Beispielen.

5.3 Grundbegriffe der Objektorientierung

Die Idee objektorientierte Software zu entwickeln, ist schon über 25 Jahre alt. Bücher über Design- und Analysemethoden erschienen jedoch erst Anfang der 90er Jahre. Beginnend mit der Programmiersprache *Simula (ca. 1970)* über *Smalltalk-76* und *C++ (ca. 1985)* ist zur Zeit *Java (ca. 1996)* die neueste Entwicklung objektorientierter Progammiersprachen.
Grundgedanke des objektorientierten Ansatzes ist es, reale Gegenstände als *Objekte* zu betrachten, die sich wiederum aus Objekten zusammensetzen. Dabei entsprechen diese nur in der Form den realen Gegebenheiten, als sie deren wesentliche, im Modell benötigten Verhaltensweisen und Merkmale aufweisen. Gleichartige Objekte werden in sogenannten *Klassen* zusammengefaßt, die einen Bauplan für die zu erzeugenden Objekte darstellen. Während Klassen eben nur einen Bauplan darstellen, sind Objekte die zur Laufzeit tatsächlich vorhandenen Speicherstrukturen. Die Notation der beiden bisher eingeführten Begriffe erfolgt, wie auch die aller folgenden Elemente des OO-Ansatzes in diesem Abschnitt, in der UML-Notation.

Abbildung 5.1: Klasse und Objekt in der UML-Notation

Wie in Abbildung 5.1 zu sehen, unterscheidet sich die Notation von Klassen und Objekten nur durch den bei den Objekten unterstrichenen Namen im ersten Abschnitt innerhalb des Rechteckes. Wie oben schon erwähnt, sollen die Objekte über Merkmale und ein Verhalten verfügen. Dieses wird zum einen durch sogenannte Attribute und zum anderen durch Methoden modelliert. Attribute stellen die Bestandteile einer Klasse bzw. eines Objektes dar, in denen Daten bzw. Informationen gespeichert werden. Methoden hingegen dienen der Manipulation, Ausgabe, Abfrage, etc. der Attribute. Die *Sichtbarkeit* der Daten und Methoden, d. h. wer auf sie zugreifen kann, wird durch die Sichtbarkeitskennzeichen **public** (für alle sicht- und nutzbar), **protected** (für die Klasse selbst und ihre Unterklassen), **private** (nur für die Klasse selbst), angezeigt.

Zusätzlich ist in der UML die Angabe von Klassen-Stereotypen, Merkmalen, Zusicherungen (vgl. Abschnitt 5.5.2), Attribut-Typen und Initialwerten möglich. Es besteht jedoch kein Zwang in der UML-Notation, außer dem Namen weitere Angaben zu Klassen bzw. Objekten zu machen; dies hängt vielmehr mit dem bisher erreichten Grad der Detaillierung zusammen.

5.3.1 Vererbung und abstrakte Klassen

Ein wichtiger und dem Drang des Menschen nach Klassifizierung nachgeahmter Mechanismus, wird durch die Vererbung beschrieben. Menschen versuchen in vieler Hinsicht, reale Objekte einer systematischen Ordnung zu unterwerfen. Dabei werden hierarchische Strukturen gebildet, die von der Spitze zur untersten Ebene eine *Spezialisierung* bzw. in umgekehrter Richtung eine *Generalisierung* darstellen. Ein an oberster Stelle stehendes Objekt verfügt dann über Merkmale und Verhaltensweisen, die alle Objekte bis zu denen der untersten Ebene ebenfalls besitzen. Auf jeder Stufe der Spezialisierung haben die Objekte jedoch zusätzliche Merkmale und Verhaltensweisen.
In der Objektorientierung ist durch dieses Prinzip die Wiederverwendungsmöglichkeit gegeben. Ferner spricht man bei Klassen, die weiter oben in der Hierarchie stehen und Merkmale bzw. Verhaltensweisen weitergeben, von *Oberklassen* , eine erbende Klasse wird *Unterklasse* genannt. Notiert wird dieses Verhältnis mit einem Pfeil von der Unter- zur Oberklasse. Die Einteilung wird häufig mittels eines mit in der UML *Diskriminator* bezeichneten Unterscheidungsmerkmales für die Unterklassen getroffen.

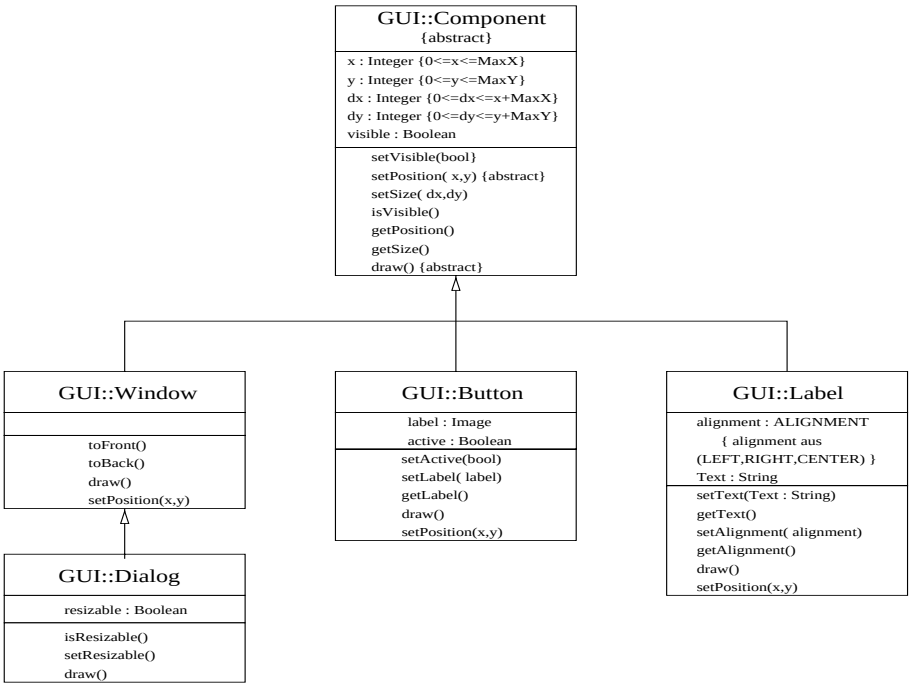


Abbildung 5.2: Klassenhierarchie in der UML-Notation

Abbildung 5.2 zeigt den Ausschnitt einer Klassenhierarchie am Beispiel einer möglichen Bibliothek zur Programmierung von grafischen Benutzeroberflächen *(GUI)*. Neben der bisher geschilderten *Einfachvererbung* ist auch die *Mehrfachvererbung* möglich. In diesem Fall hat eine Unterklasse mehr als nur eine Oberklasse.
Zwei bisher nicht erwähnte Mechanismen, die in Abb. 5.2 verwendet werden, sollen nun erläutert werden. Der Erste verbirgt sich hinter dem Merkmal $\{abstract\}$, unter dem Namen der Klasse *GUI::Component*. Dieses bezeichnet eine Klasse, von der keine Objekte erzeugt werden können. Diese Klassen werden *abstrakte Klassen* genannt und dienen dazu, die Eigenschaften, die den Unterklassen gemein sind, zu abstrahieren. Abstrakte Klassen dürfen auf allen Stufen der Hie-rarchie vorkommen. Sie können sowohl von ebenfalls abstrakten, als auch von konkreten Klassen Unter- bzw. Oberklassen sein. Eine mit $\{$abstract$\}$

gekennzeichnete Methode gibt an, daß keine konkrete Implementierung in dieser Klasse erfolgt, so daß die Unterklassen die Methode realisieren müssen. Diese Kennzeichnung ist nur innerhalb von abstrakten Klassen möglich. Der zweite Mechanismus, der durch den Zusatz *GUI::* vor dem Klassennamen angezeigt wird, beschreibt die Möglichkeit, Klassen zu *Paketen* zusammenzufassen. Pakete erlauben es, eine Menge von Klassen in kleine überschaubare Einheiten zu gruppieren, z. B. Oberflächenklassen, Datenbankinterface, etc.

5.3.2 Behälterklassen

In Bezug auf das Beispiel in Abb. 5.2 wird klar, daß noch eine Möglichkeit fehlt, mehrere grafische Objekte gleichzeitig auf dem Bildschirm darzustellen, die dabei einheitlich verwaltet werden. Für solch einen Zweck bieten sich sogenannte *Behälterklassen* an, die je nach Organisationsform Objekte in sequentiellen oder assoziativen Strukturen verwalten. Letztere ermöglichen durch zusätzliches Abspeichern eines Schlüssels zu jedem Objekt eine Identifikation. Behälterklassen unterscheiden sich im wesentlichen durch verschiedene Formen der Objektverwaltung und insbesondere der erlaubten Anzahl gleicher und unterschiedlicher Klassen.

Das Vorhandensein fertiger Behälterklassen ist programmiersprachenabhängig, jedoch stehen meistens solche Klassen wie z. B. *Array*, *String* und *Vector* zur Verfügung.
Eine Behälterklasse zur Aufnahme von grafischen Objekten, die sich in das oben beschriebene Beispiel einfügen läßt, könnte auf verschiedene Arten realisiert werden. Die in Abb. 5.3 gewählte Variante verwendet die Mehrfachvererbung, bei der einzelne Klassen sowohl die Eigenschaften der Klasse *GUI::Component*, als auch die der Behälterklasse erben.

Abbildung 5.3: Die Klasse *GUI::Container* und ihre Einordnung in die Hierarchie

5.3.3 Kommunikation zwischen Objekten

Unter der Kommunikation von Objekten versteht man das Aufrufen von Methoden der Objekte untereinander. Die Objekte sollen damit befähigt werden, die für sie vorgesehenen Aufgaben wahrzunehmen. Dazu ist es allerdings notwendig, daß sich ihre entsprechenden Klassen kennen. Bei den oben genannten Vererbungsbeziehungen ist das immer dann der Fall, wenn eine Unterklasse Methoden einer übergeordneten Klasse benutzen muß, nicht jedoch, wenn keine Vererbungsbeziehungen bestehen. Sollen die Objekte dennoch miteinander kommunizieren, so gibt es die Möglichkeit, die Zusammenarbeit durch *Assoziationen* und *Aggregationen* zu erreichen.

Assoziationen beschreiben Beziehungen zwischen Objekten, die Exemplare unterschiedlicher Klassen sein können. Während man bei Verbindungen zwischen Klassen von Assoziationen spricht, werden Beziehungen zwischen Objekten dieser Klassen Objektverbindungen oder engl. *link* genannt. In der UML wird durch eine Verbindungslinie zwischen den beteiligten Klassen oder Objekten diese Beziehungen verdeutlicht. An den jeweiligen Enden wird die Multiplizität, die die Anzahl der jeweils verbundenen

Objekte angibt, notiert. Des weiteren erhält die Verbindung einen sprechenden Bezeichner, der mit einem kleinen leserichtungsangebenden Pfeil versehen ist und das Verhältnis zwischen den Objekten beschreibt. Neben ungerichteten Verbindungen können auch gerichtete verwendet werden, wenn ein Teilobjekt nicht wissen muß, wer es benutzt.

Die Aggregation stellt eine spezielle Form der Assoziation dar. Sie verdeutlicht sogenannte „Hat-Beziehungen“, also die Zusammensetzung eines Objektes aus mehreren Einzelteilen. In diesem Zusammenhang spricht man auch von *Teile-Ganzes-Hierarchie*. Gekennzeichnet wird eine solche Beziehung in der UML durch eine Raute am Vebindungsende auf der Seite des Ganzen. Sind die Teile des Ganzen von der Existenz des Ganzen abhängig, so liegt eine *Komposition* vor. In diesem Fall werden alle Teile gelöscht, sobald auch das Ganze gelöscht wird. Es ist deshalb nicht möglich, daß ein Teilobjekt existenzabhängig von verschiedenen Aggregaten ist. Bei der Komposition wird in der UML die Raute am Ende der Verbindungslinie gefüllt, und die Seite erhält die Multiplizität 1 bzw. 0..1.

Eine wesentliche Eigenschaft von Aggregationen ist, daß das unabhängige Objekt Operationen an seine abhängigen Teilobjekte propagiert, also stellvertretend für diese handelt.

Da die Klasse *GUI::Dialog* die Eigenschaften der Klasse *GUI::Container* erbt, kann man einem Objekt dieser Klasse eine Überschrift bzw. Beschriftung geben, indem ein Objekt der Klasse *GUI::Label* hinzugefügt wird. Geht man ferner davon aus, daß ein Objekt dieser Klasse stets mindestens ein Objekt der Klasse *GUI::Button* haben muß, z. B. einen „OK“- oder „CLOSE“- Button, so ergeben sich folgende Kompositionsbeziehungen:

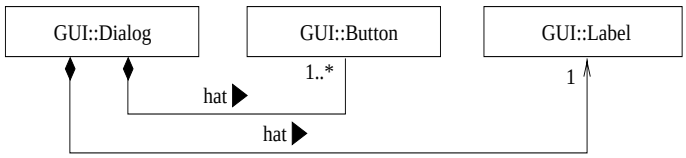


Abbildung 5.4: Kompositionsbeziehungen

Die Abbildung 5.4 zeigt, daß stets genau eine Beschriftung und mindestens ein Button existentiell von einem Dialogfenster abhängen. Während ein Button wissen sollte, wem es angehört, damit Zustandsänderungsmeldungen sofort weitergeleitet werden können, muß eine Beschriftung dieses Wissen nicht haben. Ausgedrückt wird dieses durch einen ungerichteten (Button) bzw. gerichteten Pfeil zu der betreffenden Klasse.

Die Kommunikation von Objekten besteht nun darin, daß Nachrichten ausgetauscht werden. Nachrichten betreffen stets Methoden, so daß ein Objekt nur die Nachrichten versteht, zu denen es Methoden bietet. In der zuletzt eingeführten Erweiterung des bisherigen Beispieles ist es notwendig, der Klasse *GUI::Dialog* eine eigene Methode *setPosition(x,y)*, die unabhängig von der geerbten Methode arbeitet, hinzuzufügen. Die hierzu benötigte *dynamische Polymorphie* wird in Abschnitt 5.3.4 behandelt. Diese Methode soll neben dem Setzen der eigenen Position auch die der abhängigen Elemente (Button, Label) setzen, da diese eine feste Position innerhalb des Dialogfensters haben sollen. In Abbildung 5.5 wird nur das Setzen der neuen Position bzw. der Versand der entsprechenden Nachrichten an die abhängigen Objekte notiert. Zustandsänderungsmeldungen des Button und Nachrichten an Objekte der Klasse *GUI::Label* werden zunächst außer acht gelassen.

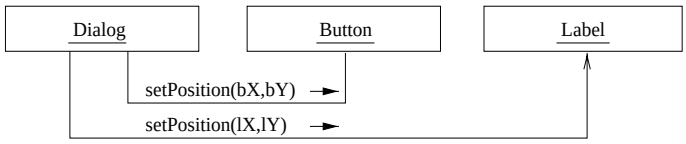


Abbildung 5.5: Nachrichten

5.3.4 Polymorphie

Die Polymorphie beschreibt einen Mechanismus der Objektorientierung, der es Klassen innerhalb von Hierarchien erlaubt, daß sich gleichnamige Methoden in unterschiedlichen Klassen unterschiedlich verhalten. Man unterscheidet zwischen *statischer* und *dynamischer* Polymorphie.

Die statische Polymorphie tritt auch bei nicht objektorientierten Programmiersprachen in Form von mathematischen Operatoren wie z. B. + oder – auf. Die Syntax dieser Operatoren unterscheidet sich hinsichtlich der unterschiedlichen Datentypen (Integer, Real, etc.) nicht. Objektorientierte Sprachen erlauben dieses Verhalten nicht nur bei den vordefinierten, sondern auch bei selbstdefinierten Datentypen. Dieses Vehalten kann auf beliebige Methoden übertragen werden und ist nicht auf die oben genannten Operatoren beschränkt.
Eine weitere Möglichkeit, die zur statischen Polymorphie zählt, ist das Definieren von Methoden gleichen Namens mit unterschiedlicher Schnittstelle. Das Verhalten dieser Methoden sollte nach außen natürlich gleich sein, unterschiedlich ist jedoch die Anzahl bzw. die Art der Parameter.

Voraussetzung für die dynamische Polymorphie ist, daß erst zur Laufzeit eines Programmes eine Zuordnung zwischen Nachricht und empfangender Methode dynamisch hergestellt wird. Da es in objektorientierten Sprachen erlaubt ist, geerbte Methoden in einer Klasse neu zu definieren (überschreiben), kann bei der Programmübersetzung noch nicht feststehen, aus welcher Klasse die Methode stammt, die eine Nachricht anspricht.
In Abb. 5.3 wird die Klasse *GUI::Container* vorgestellt, die befähigt ist, eine beliebige Anzahl von grafischen Objekten aufzunehmen. Sie besitzt Methoden zum Hinzufügen, Löschen und zum Zeichnen aller enthaltenen Objekte. Letzteres wird durch die Methode *drawAll()* bewerkstelligt, die jeweils die Methode *draw()* der grafischen Objekte aufruft. Zur Zeit der Programmübersetzung kann natürlich noch nicht feststehen, wieviele, und vor allem, welche Objekte sich zur Programmlaufzeit im Behälter befinden. Aus diesem Grunde wird erst bei Aufruf der Methode ermittelt, welche Operation (welcher Klasse) angesprochen wird.

5.4 Objektorientierte Softwareentwicklung

Als man begann, eigenständige Methoden für die objektorientierte Softwareentwicklung zu entwerfen, wurde versucht, die Vorteile der schon älteren und bewährten Methoden auf die objektorientierten zu übertragen, überwiegend auf Basis der Sprachen *Smalltalk* und *C++*. Die bekanntesten OO-Methoden sind:

- **OOA/OOD** (Object–Oriented Analysis / OO Design) von P. Coad, Ed. Yourdon
  Die in [PC91a] bzw. [PC91b] beschriebenen Methoden zeichnen sich vor allem durch gute Strukturierungsmöglichkeiten und leichte Erlernbarkeit aus.

- **OOAD** (Object—Oriented Analysis and Design) von Grady Booch
  Diese Methode läßt im Vergleich zur OOA/OOD die Modellierung von zeitabhängigem Verhalten durch Zeitdiagramme zu. Des weiteren lassen sich Objektverbindungen beschriften und Schnittstellen zwischen Teilsystemen genau beschreiben (vgl. [Boo94]).
- **OMT** (Object Modeling Technique) von Jim Rumbaugh et al.
  Im Vergleich zur OOA/OOD kann das dynamische Verhalten eines Systems besser beschrieben und das Modell simuliert werden (vgl. [Rum91]). Diese Methode lehnt sich jedoch sehr stark an die strukturierten Methoden der vor–objektorientierten Zeit an.
- **OOSE** (Object–Oriented Software Engineering) von Ivar Jacobson
  Wichtigste Neuerung in dieser Methode war die Einführung der *Use Cases* (Anwendungs-fälle), die das Zusammenwirken von Akteuren (Personen/andere Systeme) in einem System beschreibt (vgl. [Jac92]).
- **OOSA** (Object–Oriented Systems Analysis) von Shlaer und Mellor
  Es werden im wesentlichen Zustandsübergangsdiagramme und Datenflußdiagramme zur Beschreibung der statischen und dynamischen Teile des Systems eingesetzt (vgl. [SS98]).

Anfang der 90er Jahre hatten sich die Methoden von Grady Booch und James Rumbaugh zu den beliebtesten entwickelt. Die Zusammenarbeit von Booch und Rumbaugh ab 1995 führte dann zunächst zur *Unified Method (UM)* und später unter Mitwirkung von Jacobson zur *Unified Modeling Language (UML)*. Allerdings stellt die UML lediglich die Beschreibung einer einheitlichen Notation und die Definition eines *Metamodells*, also eines formalen Modells, mit dem sich Modelle entwickeln und erweitern lassen, dar, während die Beschreibung einer Entwicklungsmethode ausgeklammert wird. Eine Entscheidung für eine dem zu entwickelnden System angepaßte Methodik muß also stets gesondert getroffen werden. Im folgenden wird eine in [Oes97] beschriebene Methode erläutert, die sich dem Autor nach in der Praxis bewährt hat.

5.4.1 Das Vorgehensmodell

Die objektorientierte Enwicklung von Softwaresystemen läßt sich durch ein iteratives und evolutionäres Vorgehensmodell beschreiben, das durch eine Gliederung der Aktivitäten während der Entwicklung und die Verzahnung dieser Aktivitäten unterschiedlichen Detaillierungsgrades, gekennzeichnet ist. Durch das iterative Vorgehen wird jedoch nicht festgelgt, daß auf jeder Iterationsstufe für alle Bereiche des Systemes auf der gleichen Detaillierungsebene entwickelt wird. Vielmehr wird der Grad der Detaillierung, je nach Problemstellung, Priorität und Anforderung an z. B. Laufzeitverhalten oder Nebenläufigkeit, angepaßt.

Zu Beginn werden die grundlegenden Systemanforderungen in einem Initial–Workshop festgelegt (RAD – Rapid Analysis & Design). Diese Phase ist durch eine sehr enge Zusammenarbeit zwischen EntwicklerInnen, AnwenderInnen und AuftraggeberInnen geprägt. Betrifft das Thema des Projektes gleich mehrere Anwendungsgebiete z. B. Fachabteilungen, so kann, falls erforderlich, auch für jede dieser ein solcher Workshop durchgeführt werden. Ziel ist eine erste schnelle Analyse und ein Prototyping. Für die EntwicklerInnen ist es in dieser Zeit wichtig, sich die Begriffe und Kontexte der betreffenden Anwendungsumgebung zu erarbeiten und Materialien (z. B. Rechnungen, Verträge, Formulare, etc.) zu sammeln. Dazu wird zunächst mit der Analyse der Geschäftsprozesse begonnen, um zu ermitteln, welche Anwendungsfälle auftreten. Anwendungsfälle beschreiben typische Interaktionen zwischen AnwenderInnen und System, die nötig sind, um einen Arbeitsgang zu erledigen. Anwendungsfälle sollten dabei nicht

zu umfangreich werden und sich nur auf Interaktionen innerhalb eines kleinen Zeitintervalles beziehen. Es werden ein Fachlexikon (Glossar) mit der Beschreibung der Anwendungsfälle, sowie Anwendungs-(fragment-)prototypen erstellt. Die Auftraggeber können sich ein erstes Bild von den Entwicklungen machen. Zudem werden, für die zukünftige Zusammenarbeit wichtig, die Kommunikationswege, -partner und -strukturen verfestigt. Mit einer Bewertung und Kritik der bisherigen Ergebnisse durch alle Beteiligten endet diese erste Phase.
Das weitere Vorgehen bzw. die Planung der Evolutionszyklen wird duch das Projektmanagement auf Basis der Workshop-Ergebnisse festgelegt. Ein solcher Evolutionszyklus besteht dabei aus den folgenden vier Phasen (siehe Abschnitte 5.4.2 – 5.4.5):

- **Analyse**
- **Design**
- **Implementierung**
- **operativer Einsatz**

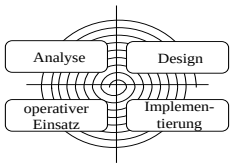


Abbildung 5.6: 4–Phasen–Zyklus

Der Zyklus endet mit einer Überprüfung der Ergebnisse (Review). Anschließend beginnt ein neuer Zyklus. Die genaue Abfolge der vier Phasen innerhalb eines Zyklus ist nicht starr, sie wird vielmehr der Situation ensprechend individuell angepaßt. Sowohl diese vier Phasen als auch der gesamte Prozeß der objektorientierten Systementwicklung verläuft spiralförmig von innen nach außen und zeigt die fortschreitende Entwicklung des geforderten Systems.

5.4.2 Analyse

Zu Beginn der Analysephase wird eine Geschäftsprozeßanalyse durchgeführt. Sie soll Aufschluß darüber geben, welche Teile des Systems sich gegebenenfalls durch Software automatisieren oder unterstützen lassen. Dabei können sich Optimierungsmöglichkeiten ergeben, die sich durch einfache organisatorische Umstrukturierungen verwirklichen lassen.

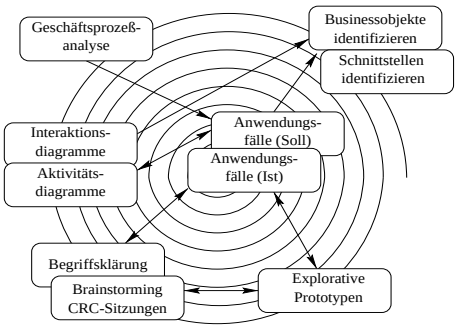


Abbildung 5.7: Analyse–Phase

Die sich ergebenden Systemrelevanten und durch Software zu unterstützenden bzw. automatisierenden Abläufe müssen nun genau untersucht und in Anwendungsfällen beschrieben werden. Man unterscheidet dabei zwischen den Regel– und Sonderfällen, die in primäre und untergeordnete sekundäre Anwendungsfälle gegliedert werden. Es entsteht ein Abbild des Ist–Zustandes. Anwendungsfälle dienen vor

allem auch dazu, den SoftwareentwicklerInnen ein genaues Verständnis der Materie zu vermitteln.

Auf Basis des Ist-Zustandes wird nun ein Soll–Zustand erarbeitet. Dazu wird zu jedem Ist– ein Soll–Anwendungsfall erarbeitet und somit die zukünftigen Arbeitsabläufe formuliert. Erste Prototypen sind inbegriffen, um den AuftraggeberInnen und AnwenderInnen einen Einblick und die Möglichkeit zu geben, frühzeitig Kritik zu üben. Solche Prototypen sind meist auf Dialoge bzw. Eingabemasken beschränkt. Das Erschließen und Festhalten der Begriffe der Anwendungswelt in einem Glossar gehört stets zur permanenten Tätigkeit der EntwicklerInnen.

Elementare Begriffe (Orte, Personen, Gegenstände, Konzepte oder Dokumente) des realen Ge-schäftslebens werden zunächst als Businessobjekte bestimmt, ihre Schnittstellen werden durch die ermittelten Verantwortlichkeiten identifiziert. Erste Aggregationsbeziehungen ergeben sich aus den Beziehungen bzw. der Zusammenarbeit der Businessobjekte. Beispiele für Businessobjekte sind: Kunde, Reservierung, Rechnung, Kfz, etc. Gemein ist ihnen jedoch, daß ihr Detaillierungsgrad auf einem solchen Niveau gehalten wird, der z. B. Fachabteilungen und Managern ein leichtes Verständnis erlaubt.
Eine gute Möglichkeit, die Begriffe der Anwendungswelt zu erfassen und einen umfassenden Überblick zu erlangen, sind z. B. Brainstormings mit **CRC–Karten (Class–Responsibilities–Collaborators)** bzw. Klassenkarten und **Mind–Maps**. Auf solchen CRC–Karten werden die Klasse, ihre Verantwortlichkeiten und andere Beteiligte notiert. Als Verantwortlichkeiten einer Klasse werden alle Operationen und das Wissen, welches ihre Objekte später benötigen, definiert. Eine Klasse muß in vielen Fällen mit anderen Klassen kooperieren. Daher werden Beteiligte als solche Klassen bezeichnet, die zur Erfüllung der Verantwortlichkeiten einer Klasse benötigt werden. Eingesetzt werden die CRC–Karten vor allem zur Strukturierung und Detaillierung der Begriffe aus dem Anwendungsbereich. Jeder Anwendungsfall wird so nach Klassen durchgearbeitet, und es entsteht eine Klassensammlung, die auf CRC–Karten notiert wird. Anschließend dienen diese Karten als Diskussionsgrundlage innerhalb der CRC–Workshops, in denen durch zum Teil unterschiedliche Ansichten über die Anwendungswelt neue Klassen, Verbindlichkeiten oder Beteiligte identifiziert werden oder sich als überflüssig herausstellen.
Mind–Maps hingegen stellen vor allem dem Analytiker eine gute Möglichkeit bereit, für sich selbst Skizzen der beobachteten, gelernten oder erfragten Abläufe innerhalb der Anwendungswelt zu machen. Die Technik, die die kreativen Möglichkeiten des menschlichen Gehirnes aktivieren soll, wird z. B. in [Bey93] näher beschrieben.

5.4.3 Design

In der Design–Phase muß ein abstraktes Modell aus den Begriffen, Sachverhalten und Vorgängen der Anwendungswelt erschaffen werden. Dieses Modell, das den in der Analyse–Phase erkannten Anforderungen gerecht werden muß, dient dann als Vorlage für die zu implementierende Software. Zudem ist auf leichte Änderbarkeit des Modelles zu achten, da eine Änderung in den Anforderungen im Modell leicht nachvollziehbar sein soll.
Das entstehende Modell enthält:

- **Basiselemente**
  z. B. Klassen, Objekte, Attribute, Operationen bzw. Methoden, Zusicherungen, Merkmale
- **statische Elemente**
  z. B. Assoziationen, Aggregationen, Vererbungsbeziehungen, Rollen

- **dynamische Elemente**
  z. B. Aktivitäts– und Zustandsdiagramme, Sequenz– und Kollaborationsdiagramme.

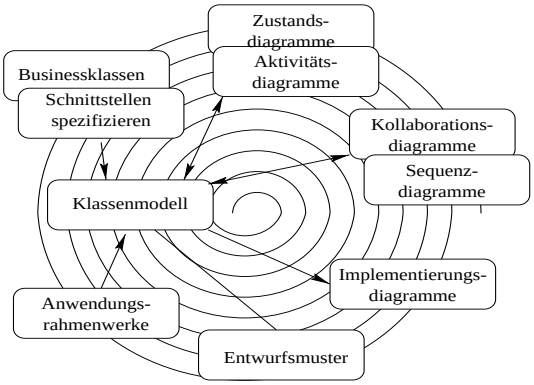


Abbildung 5.8: Design–Phase

Bevor jedoch mit dem eigentlichen Design begonnen wird, sollte eine Anwendungsarchitektur festgelegt werden. Eine solche Architektur legt den inneren Aufbau bzw. die Struktur der Anwendungsbausteine bzw. Pakete fest und bestimmt darüberhinaus, welche Klassen und damit verbunden, welche Schnittstellen definiert werden müssen. Sie ist Hilfsmittel für eine sinnvolle Arbeitsteilung und Übersichtlichkeit des Projektes, verspricht eine langfristige Flexibilität während der Entwicklung und hilft, einen höheren Wiederverwendungsgrad zu erreichen. Ein möglicher Rahmenaufbau eines Anwendungsbausteines könnte folgende Klassen (–Gruppen) enthalten:

- **Dialogsteuerung**
  Sie umfaßt alle Klassen, die zur Präsentation und Eingabe von Informationen bzw. Daten benötigt werden. Eine inhaltliche Verarbeitung der Daten findet hier jedoch nicht statt. Sobald eine Eingabebestätigung durch den Anwender erfolgt ist, werden alle eingegebenen Daten sowie die Kontrolle an die Interaktionssteuerung weitergegeben.

- **Interaktionssteuerung**
  Diese Schicht bzw. ihre Klassen interpretieren die von der Dialogsteuerung kommenden Daten nur soweit, wie es zur Bestimmung der Bearbeitungssituation notwendig ist. Sie ist verantwortlich für die Kommunikation zwischen den Dialog– und Fachklassen und entkoppelt diese voneinander.

- **Vorgangssteuerung**
  Die Klassen dieser Schicht sind für alle fachlichen Vorgänge, die bei der Abarbeitung eines Geschäftsvorfalles notwendig sind, verantwortlich und initiieren diese.

- **Business– und Fundamentalklassen**
  Diese Klassen stellen das eigentliche Modell der Anwendungswelt dar und verbergen deren Attribute, Methoden und Zusicherungen. Sie enthalten das gesamte Fachwissen.

Eine möglichst einheitliche Verwendung einer solchen Architektur sollte für die gesamte Anwendungsentwicklung stattfinden. Durch ein Rahmenwerk oder engl. *Framework*, das die Infrastruktur bzw. die Kommunikation zwischen den Schichten definiert, kann dies sichergestellt werden. Das Rahmenwerk umfaßt dabei eine Vielzahl abstrakter Klassen, von denen die konkreten Klassen abgeleitet werden. Die

Kommunikation innerhalb der Anwendung verläuft dann soweit wie möglich über die Klassen des Frameworks, die der aufgesetzten Klassen nach Möglichkeit nur innerhalb der jeweiligen Schicht.
Nachdem diese Vorentscheidungen getroffen sind, kann damit begonnen werden, die Klassen zu identifizieren. Zunächst werden alle in den Anwendungsfällen und dem Fachlexikon verwendeten Begriffe als Klassen angesehen. Anschließend wird versucht, diese Begriffe zu gruppieren und dabei Oberbegriffe und Ähnlichkeiten zu suchen.
Im Anschluß daran werden die Beziehungen zwischen den gefundenen Klassen auf Assoziationen, Aggregationen und Vererbung untersucht. Dabei sind stets die Anforderungen zu berücksichtigen, und das Modell ist dementsprechend zu gestalten.
Aus den erarbeiteten Anwendungsfällen lassen sich durch Betrachtung der beschriebenen Interaktionen zwischen den Objekten die notwendigen Operationen bzw. Methoden ermitteln. Das dabei eventuell neu erlangte Verständnis der Beziehungen zwischen den Klassen ist in das Modell einzuarbeiten. Gegebenenfalls müssen die Assoziations–, Aggregations– und Vererbungsbeziehungen geändert oder vervollständigt werden.
Neben den Operationen werden nun auch die Attribute und ihre Datentypen festgelegt. Dabei werden die Objekte hinsichtlich des erforderlichen Wissens durchleuchtet. Sind alle Attribute identifiziert, so ist zu hinterfragen, ob einige Attribute als eigenständige Objekte erfaßt werden müssen. Indizien für solche Attribute sind z. B. eine eigene Identität oder die Tatsache, daß andere Klassen ebenfalls zu diesem Attribut Beziehungen unterhalten wollen.

Folgende Diagrammtypen der UML sind in der Design–Phase weiterhin von Bedeutung:

- **Aktivitätsdiagramme** (vgl. Abschnitt 5.5.1.6)
- **Zustandsdiagramme** (vgl. Abschnitt 5.5.1.5)
  Sofern die Komplexität eines zu entwerfenden Objektes dies erfordert.
- **Kollaborations– und Sequenzdiagramme** (vgl. Abschnitt 5.5.1.3 bzw. 5.5.1.4)

5.4.4 Implementierung

In der Implementierungsphase wird das in der Design–Phase entworfene Modell schrittweise in ein lauffähiges System umgesetzt.

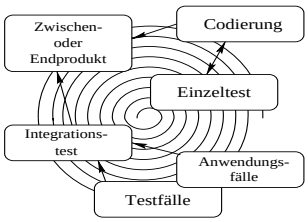


Abbildung 5.9: Implementierung

Zunächst werden einzelne Klassen ihrer Spezifikation entsprechend umgesetzt und getestet. Im nächsten Schritt werden dann Klassen zu Paketen oder Teilsystemen integriert, und das Zusammenspiel der Klassen wird getestet. Abhängig von der Größe des Produktes entsteht so das End– oder Zwischenrodukt.

Das erarbeitete System muß nun zusammen von den EntwicklerInnen und AnwenderInnen bezüglich der in der Analysephase beschriebenen Anwendungsfällen und den in der Design–Phase erstellten Interaktionsmodellen überprüft werden.

5.4.5 Der operative Einsatz

Abschließend wird das bisher erstellte Zwischen– oder Endprodukt entweder einer ausgewählten Gruppe, oder allen AnwenderInnen zum operativen Einsatz zur Verfügung gestellt. Bei einem ersten Einsatz

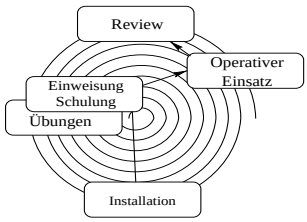


Abbildung 5.10: Operativer Einsatz

innerhalb einer kleinen Gruppe zeigt sich auch, wie detailliert der Einsatz einzelner Teilsysteme geschult werden muß. Alle Beobachtungen fließen nach ihrer Auswertung dann in den nächsten Zyklus mit ein.

5.5 Die Unified Modeling Language (UML)

Wie schon in Abschnitt 5.4 kurz beschrieben, basiert die UML auf Arbeiten von Jacobson, Rumbaugh und Booch. Die Drei, seit ihrer Zusammenarbeit bei der Firma Rational im Volksmund „Amigos“ genannt, begannen die Arbeit an der UML im Jahr 1996. Die Version 1.0 wurde im Januar 1997 bei der *Object Management Group (OMG)* eingereicht. Die Version 1.1 folgte im September desselben Jahres, woraufhin alle Gegenvorschläge zurückgezogen wurden, da deren Konzepte Berücksichtigung fanden. Die Unified Modeling Language stellt eine einheitliche, universelle Notation dar, mit der sich Modelle von Softwaresystemen spezifizieren, konstruieren, visualisieren und auch dokumentieren lassen. Durch eine Vielzahl von mächtigen Modellelementen erlaubt es die UML unter anderem, konkurrierende, verteilte und zeitkritische Systeme zu modellieren. Allerdings wird keine Methode zur Modellierung von Systemen beschrieben.
Neben den allgemeingültigen Anforderungen an Modellierungssprachen, wie z. B.

- überschaubare Anzahl von Konzepten und Symbolen und Konsistenz bei der Verwendung in allen Bereichen der Notation
- Mehraufwand bei der Modellierung nur in Ausnahmesituationen,
- Schichtenarchitektur, die Ergänzen/Weglassen von Detailreichtum ermöglicht,

spielte auch die Zeichenbarkeit der Notationselemente eine Rolle bei der Entwicklung der UML.

5.5.1 Die Diagrammtypen der UML

Die UML besitzt eine Vielzahl vom Modellelementen. Im folgenden werden die Diagrammtypen erläutert. Einige Elemente sind in verschiedenen Diagrammen enthalten. Erläutert werden sie immer dort, wo sie primär Verwendung finden.

- Anwendungsfalldiagramm (Use Case Diagram)
- Klassendiagramm (Class Diagram)
- Verhaltensdiagramme (Interaction Diagrams)
  - Kollaborationsdiagramm (Collaboration Diagram)
  - Sequenzdiagramm (Sequence Diagram)
  - Zustandsdiagramm (State Diagram/Charts)
  - Aktivitätsdiagramm (Activity Diagram)
- Implementierungsdiagramme (Implementation Diagrams)
  - Komponentendiagramm (Component Diagram)
  - Einsatzdiagramm (Deployment Diagram)

5.5.1.1 Das Anwendungsfalldiagramm

Das Anwendungsfalldiagramm (s. Abb. 5.11) zeigt die Beziehungen zwischen den in Abschnitt 5.4 erläuterten Anwendungsfällen und den daran beteiligten Akteuren/Benutzern. Da Anwendungsfälle nicht das interne Verhalten des zu entwickelnden Systemes darstellen, sondern vielmehr nur eine Abbildung der Interaktionen zwischen System und Benutzer sind, können die **Use Case Diagramme** nur einen Kontext und eine Gliederung für die Beschreibung bilden, wie mit einem Geschäftsvorfall umgegangen wird. Anwendungsfälle werden durch eine textuelle Beschreibung spezifiziert und enthalten u.a. die Akteure und Vor- und Nachbedingungen. Ein Anwendungsfalldiagramm zeigt eine Menge von Anwen-

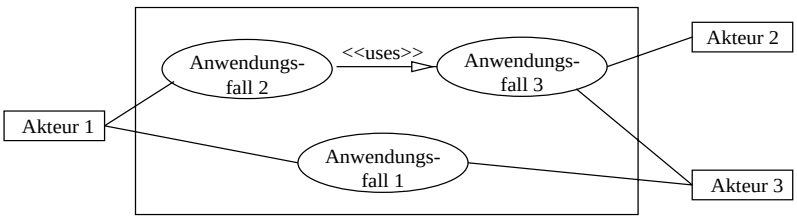


Abbildung 5.11: Anwendungsfalldiagramm

dungsfällen (durch Ellipsen gekennzeichnet), die untereinander durch Vererbungspfeile verbunden sein können. Vererbungspfeile können mit den Stereotypen (vgl. Abschnitt 5.5.2) ≪*uses*≫ oder ≪*extends*≫ beschriftet sein und sind immer von dem benutzenden zum mitbenutzten Anwendungsfall gerichtet. Linien zwischen den Anwendungsfällen und den Akteuren verdeutlichen, welcher Akteur an welchem Anwendungsfall beteiligt ist.

≪**uses**≫ wird verwendet, wenn eine Anwendungsfallbeschreibung in mehreren Beschreibungen benötigt wird. So können mehrfach benötigte Teile separiert und wiederverwendet werden.

≪**extends**≫ wird verwendet, um Variationen des Normalfalles, also Ausnahme- bzw. Fehlerfälle, aber auch spezielle Abweichungen und Erweiterungen des Standardfalles anzuzeigen. Der Pfeil zeigt dann von der Variante zum Normal–Anwendungsfall.

Ein Rechteck um eine Menge von Anwendungsfällen zeigt die Systemgrenzen. Eine Hierarchisierung innerhalb dieses Diagrammtypes ist dadurch möglich, daß einzelne Anwendungsfälle durch ein Rechteck umrandet und an anderer Stelle durch ein eigenes Diagramm beschrieben werden.

5.5.1.2 Das Klassendiagramm

Das Klassendiagramm (s. Abb. 5.2 und Abb. 5.12) zeigt die statische Struktur eines modellierten Systems. Beschrieben werden die Struktur der Klassen innerhalb des Systems, die Beziehungen der Klassen untereinander und die Attribute und Methoden der einzelnen Klassen. Ein Klassendiagramm kann neben Klassen auch Objekte enthalten. Sind nur Objekte enthalten, so spricht man von einem Objektdiagramm, das eine Momentaufnahme zur Laufzeit des Systems darstellt.
Die nachfolgende Abbildung zeigt die wichtigsten Elemente des Klassendiagrammes, die zum Teil auch schon in Abschnitt 5.3 in den Abbildungen 5.1 – 5.5 verwendet wurden. Bisher nicht erwähnt wurde der Begriff der Schnittstelle. Schnittstellen (engl. Interfaces) beschreiben einen ausgewählten Teil des nach außen sichtbaren Verhaltens von Modellelementen (insbesondere von Klassen) oder einer Menge davon. Sie beinhalten Signaturen von Operationen, zu welchen die Elemente, die diese Schnittstelle anbieten wollen, eine Implementierung besitzen müssen. Schnittstellenklassen werden mit dem Stereotyp ≪*interface*≫ versehen.

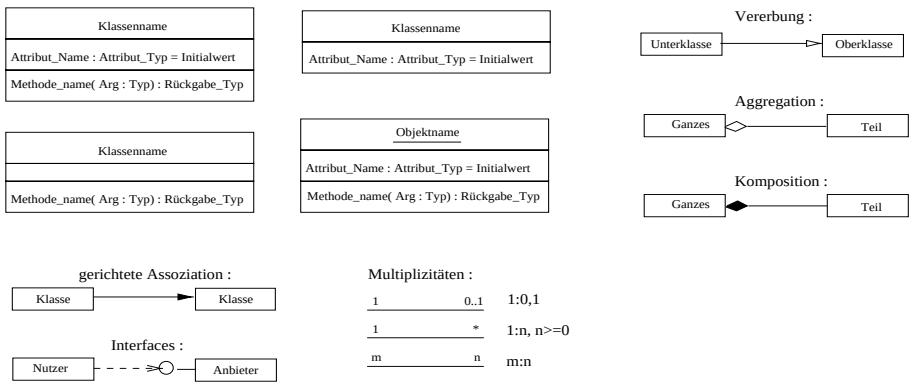


Abbildung 5.12: Einige Elemente des Klassendiagrammes

5.5.1.3 Das Kollaborationsdiagramm

Unter einer Kollaboration versteht man eine Menge von Interaktionen zwischen ausgewählten Objekten in einer bestimmten begrenzten Situation (Kontext). Die Betonung bei der Darstellung einer Kollaboration in einem Kollaborationsdiagramm (s. Abb. 5.13) liegt auf den Beziehungen zwischen den Objekten und ihrer Topographie zueinander.
Ein Kollaborationsdiagramm zeigt ausgewählte Nachrichten zwischen den Objekten und wie diese den

Eingang der Nachrichten interpretieren. Der zeitliche Verlauf der Kommunikation wird durch Numerierung der Nachrichten verdeutlicht. Neben der zeitlichen Abfolge der Nachrichten werden im Kollaborationsdiagramm die Namen der Nachrichten und Antworten, sowie ihre möglichen Argumente dargestellt. Damit zwei Objekte miteinander kommunizieren können, muß das sendende Objekt eine Referenz auf den Empfänger der Nachricht besitzen. Es muß also eine Assoziation bestehen. Diese muß jedoch nicht grundsätzlich, sondern kann auch temporär ,z. B. als Argument übergeben, vorhanden sein. Ein Objekt kann jedoch stets mit sich selbst kommunizieren.
Die Nachrichten werden auf Assoziationslinien notiert, die zwischen den Objekten dargestellt sind. Ein kleiner Pfeil an der Nachricht zeigt immer vom Sender zum Empfänger. Mögliche Argumente einer Nachricht werden ebenfalls notiert. Antworten werden in der Notation *antwort:=nachricht()* dargestellt. Die allgemeine Syntax der Nachrichtenbezeichnung und die Bedeutung ihrer Bestandteile bezeichnet der folgende Ausdruck.

[Vorgänger-Bedingung] Sequenzausdruck Antwort:= Nachrichtenname(Parameterliste)

Vorgänger-Bedingung Die Vorgänger-Bedingung besteht aus einer Aufzählung von Sequenznummern notwendigerweise vorher gesendeter Nachrichten, d. h. diese Nachrichten müssen also gesendet worden sein, bevor diese Nachricht gesendet werden darf. Somit kann eine Synchronisation herbeigeführt werden. Die Angabe der Vorgänger-Bedingung ist optional.

Sequenzausdruck Eine aufsteigende Numerierung verdeutlicht die Reihenfolge der Nachrichten. Werden innerhalb einer Operation, die eine empfangene Nachricht interpretiert, neue Nachrichten gesendet, so erhalten diese eine durch einen Punkt getrennte neue Sequenznummer. Iterationen werden durch '*' gekennzeichnet. Das Ende einer Sequenznummer wird durch einen Doppelpunkt angegeben. Ferner ist die Angabe einer Bedingung möglich, die das bedingte Versenden von Nachrichten ermöglicht.

Antwort Eine von einer Nachricht gelieferte Antwort kann einen Namen haben, der wiederum in anderen Nachrichten als Argument verwendet werden kann. Er kann der Name einer lokalen Variable oder eines Objektattributes sein.

Nachrichtenname(Parameterliste) Der Nachrichtenname entspricht i. d. R. der Signatur einer gleichlautenden Operation, die die Nachricht interpretiert.

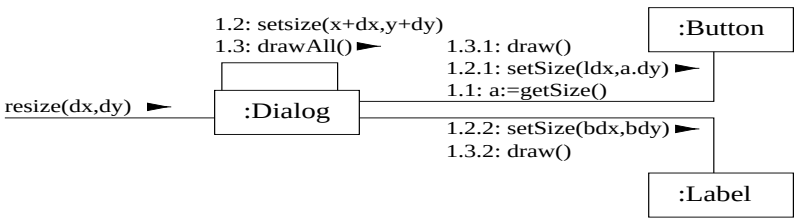


Abbildung 5.13: Kollaborationsdiagramm

In Kollaborationsdiagrammen können Stereotypen (vgl. Abschnitt 5.5.2) vorkommen. Einige von diesen beziehen sich auf die Lebenszeit von Objekten. Mit ≪*new*≫ werden Objekte gekennzeichnet, die innerhalb des dargestellten Szenarios erzeugt werden. ≪*destroyed*≫ hingegen bezeichnet Objekte, die zerstört werden und ≪*transient*≫ solche, für die beides gilt.

Weitere Stereotypen verdeutlichen die Ursache für Beziehungen zwischen Objekten, die Grundlage für den Nachrichtenaustausch sind. Folgende sich selbst erklärende Stereotypen kommen dafür in Frage: ≪*association*≫, ≪*global*≫, ≪*local*≫, ≪*parameter*≫, ≪*self*≫.

5.5.1.4 Das Sequenzdiagramm

Sequenzdiagramme (s. Abb. 5.5.1.4) beschreiben genau wie Kollaborationsdiagramme Szenarien, in denen Objekte über Nachrichten miteinander kommunizieren. Jedoch liegt hier die Betonung auf dem zeitlichen Aspekt des Nachrichtenversandes.
Die Objekte werden durch Objektsymbole dargestellt, unter denen senkrechte Lebenslinien den zeitlichen Verlauf der Nachrichten hervorheben. Die Zeit verläuft dabei von oben nach unten. Durch Überlagerung der Lebenslinien durch senkrechte nichtausgefüllte Balken wird der Steuerungsfokus, der anzeigt, welches Objekt gerade die Programmkontrolle hat, symbolisiert.

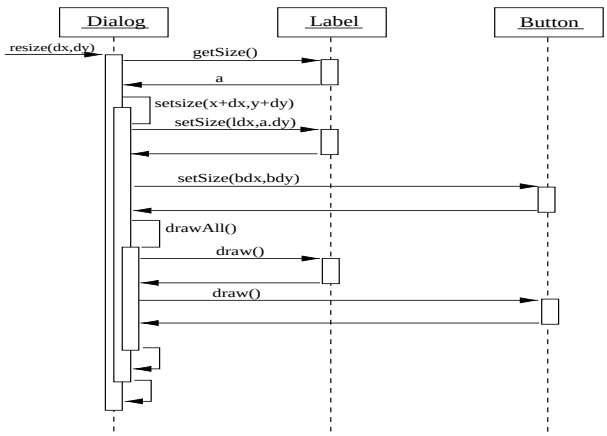


Abbildung 5.14: Sequenzdiagramm

In Sequenzdiagrammen kann auch das Erzeugen und Entfernen von Objekten dargestellt werden. Während die Konstruktion durch eine in Richtung Objektsymbol gehende Nachricht angezeigt wird, wird die Destruktion eines Objektes durch ein Kreuz am Ende des Steuerungsfokus dargestellt. Zu beiden Seiten des Diagrammes können zudem frei formulierte Erläuterungen und Zeitanforderungen notiert werden (siehe auch *Zusicherungen* in Abschnitt 5.5.2).

5.5.1.5 Das Zustandsdiagramm

Zustände gehören jeweils zu genau einer Klasse. Ein Zustand beschreibt eine Zusammenfassung von möglichen Attributwerten eines Objektes der Klasse. Nicht jede Änderung eines Attributwertes führt auch zu einer Zustandsänderung. Vielmehr wird soweit abstrahiert, daß nur die zu einem geänderten Verhalten führenden Änderungen zu einem neuen Zustand führen.
Ein Zustandsdiagramm (s. Abb.5.15) zeigt eine Folge von Zuständen, die ein Objekt innerhalb seiner Lebenszeit einnehmen kann. Ferner werden alle Stimuli (Ereignisse), die zu einer Zustandsänderung führen, gezeigt. Stimuli können an Bedingungen geknüpft sein.
Zustandsdiagramme in der UML basieren auf den von D. Harel eingeführten Statecharts. Diese sind

Zustandautomaten, die um einige Komponenten erweitert wurden. Detailierte Erläuterungen zur Syntax von Statecharts finden sich in [Har87].

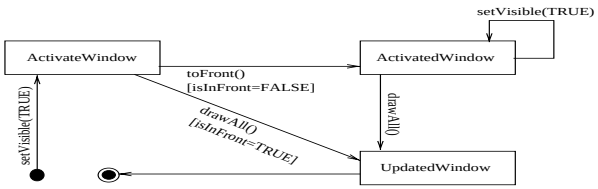


Abbildung 5.15: Zustandsdiagramm

5.5.1.6 Das Aktivitätsdiagramm

Ein Aktivitätsdiagramm (s. Abb. 5.16) ist eine spezielle Form des Zustandsdiagrammes. Sie stellt eine Mischung aus verschiedenen bisher bekannten Darstellungstechniken dar. Grundlage waren u. a. Petrinetze und Flußdiagramme. In einem Aktivitätsdiagramm sind überwiegend Aktivitäten enthalten, Zustände können zur Kennzeichnung von Abhängigkeiten hinzugefügt werden. Aktivitäten sind Zustände, in denen interne Vorgänge ablaufen. Mindestens eine Transition erfolgt immer automatisch am Ende an einer internen Aktion. Sind mehrere Transitionen vorhanden, so sind diese zur Unterscheidung an Bedingungen geknüpft.

Aktivitäten und Aktivitätsdiagramme sind entweder einer Klasse, einer Operation oder einem Anwendungsfall untergeordnet. Beschrieben werden die möglichen internen Abläufe der Modell-elemente.

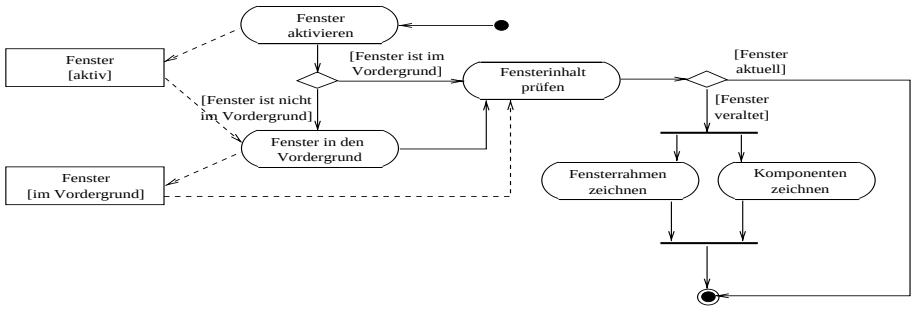


Abbildung 5.16: Aktivitätsdiagramm

In einem Aktivitätsdiagramm enthalten sind zunächst die Aktivitäten, die durch Figuren mit gerader Ober- und Unterseite, sowie konvex geformten Seiten dargestellt werden. Sie enthalten Aktionsbeschreibungen in Form des Namens, einer frei formulierten Beschreibung, Pseudocode oder Programmcode. Die sie verbindenden Transitionen, die durch Pfeile repräsentiert werden, können geteilt, aber auch synchronisiert werden. Dieses wird durch dickere Linien, von denen die Transitionen abgehen bzw. auf die sie treffen, dargestellt. Wie oben beschrieben können die Transitionen mit Bedingungen versehen werden, die in eckigen Klammern geschrieben sind. Diese müssen nicht an den die Aktivität verlassenden Transitionen stehen, sondern können in, durch Rauten dargestellten Verzweigungspunkten ausgelagert sein.

5.5.1.7 Das Komponentendiagramm

In Komponentendiagrammen (s. Abb. 5.17) können benötigte Compiler- und Laufzeit- Abhängigkeiten notiert werden. Das Komponentendiagramm wird so zu einem Hilfsmittel für das Konfigurationsmanagement. Es werden die Abhängigkeiten zwischen den Softwarekomponenten, genauer die Abhängigkeiten zwischen Quellcode, Binärcodekomponenten und ausführbaren Programmen, gezeigt. Zum Teil existieren diese Komponenenten nur zur Übersetzungszeit, andere nur zur Programmlaufzeit. Komponenten können zudem über Schnittstellen verfügen. Eine einzelne Komponente wird als Rechteck notiert. An

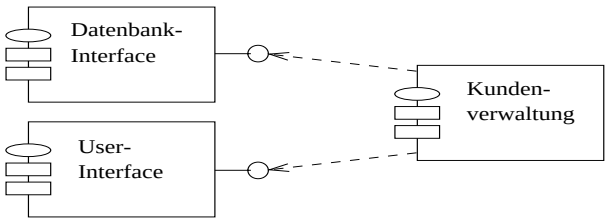


Abbildung 5.17: Komponentendiagramm

der linken Seite befinden sich zwei weitere kleinere Rechtecke, über denen eine kleine Ellipse gezeichnet wird. Neben dem Namen können innerhalb der Komponente weitere Elemente (Objekte, Komponenten, Knoten(siehe Abschnitt 5.5.1.8)) enthalten sein.

5.5.1.8 Das Einsatzdiagramm (Knoten)

Mit Einsatzdiagrammen (s. Abb. 5.18) wird modelliert, welche Komponeneten und Objekte auf welchen Knoten laufen. Es ergibt sich so ein Bild, wie die Knoten konfiguriert sind und welche Abhängigkeiten dort bestehen. Knoten sind dabei zur Laufzeit physisch vorhandene Objekte, die über Rechenleistung bzw. Speicher verfügen, also Computer (Prozessoren), Geräte, u. a. .

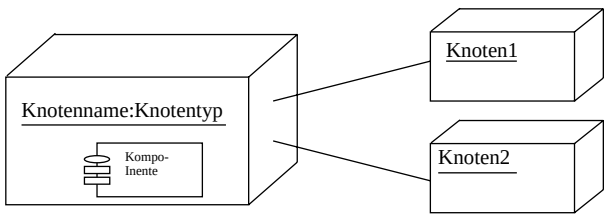


Abbildung 5.18: Einsatzdiagramm

Dargestellt werden Knoten durch Quader, die zum einen den Namen (plus Knotentyp) und zum anderen optional weitere Laufzeitobjekte (Prozesse) und Komponenten enthalten. Miteinander kommunizierende Knoten werden durch Assoziationslinien verbunden. Schnittstellen und Abhängigkeitsbeziehungen zwischen den Elementen sind ebenfalls möglich.

5.5.2 Extensionsmechanismen der UML

Die UML bietet drei Extensionsmechanismen an, mit denen die Standard- Modellelemente mit benutzerdefinierten Einschränkungen und Erweiterungen versehen werden können:

Stereotypen Sie stellen einen Mechanismus dar, die vorhandenen Modellelemente mit spezifischen und mit der Definition einer Semantik versehenen Erweiterung auszustatten. Dadurch wird das Modellelement direkt semantsich beeinflußt. Stereotypen werden in eckigen Klammern geschrieben, z. B. ≪uses≫ und vor bzw. über dem Elementnamen plaziert. Sie klassifizieren vor allem mögliche Verwendungszusammenhänge von Klassen, Beziehungen oder Paketen. Auch Attribute und Operationen können mit Stereotypen versehen werden.

Zusicherungen Mit Hilfe von Zusicherungen steht ein Mechanismus zu Verfügung, die möglichen Inhalte, Zustände oder die Semantik eines Modellelementes einzuschränken. Es können die zulässigen Wertemengen von Atrributen, Vor- und Nachbedingungen von Operationen, u. ä. beschrieben werden. Die in geschweiften Klammern geschriebenen Zusicherungen werden frei formuliert (Pseudocode, formelähnlich) oder als Merkmal, Stereotyp oder Abhängigkeit notiert. Sie fordern oder verbieten spezielle Eigenschaften und können an beliebige Notationselemente angehängt werden.

Merkmale Durch Merkmale wird die Semantik von Modellelementen um charakteristische Merkmale erweitert. Sie sind spezifische Schlüsselwort- Wert- Paare. Ähnlich wie Attribute, die die Eigenschaften einer Klasse beschreiben, spezifizieren Merkmale die Eigenschaften beliebiger Modellelemente weiter. Eine klare Trennung von Zusicherungen und Merkmalen ist nicht möglich, da z. B. die Zusicherung $\{abstract\}$ semantisch gleichbedeutend mit dem Merkmal $\{abstract{=}true\}$ ist. Merkmale, die ebenfalls in geschweiften Klammern geschrieben werden, beeinflussen im Gegensatz zu den Zusicherungen in den meisten Fällen (abhängig vom Modellierungswerkzeug) die Code- Generierung, sind also vorzuziehen.

Obwohl Merkmale und Stereotypen beliebig vergeben werden können, ist es sinnvoll, sich innerhalb eines Projektes auf eine begrenzte und wohldefinierte Menge von ihnen zu einigen. Dieses ergibt sich u. U. durch die Zusammenhänge mit der Codegenerierung von selbst.

5.6 Abschließende Bemerkungen

Die UML bietet eine Vielzahl von Diagrammtypen bei einem beachtlichen Detailreichtum. Die Modellelemente lassen sich bei Bedarf in geeigneter Weise erweitern und miteinander verknüpfen, so daß das Verständnis und die Anwendung aller Notationselemente mit allen Facetten zunächst sehr schwierig sein dürfte. Beschränkt man sich jedoch auf die grundlegenden und unbedingt notwendigen Elemente mit jeweils entsprechendem Detailreichtum, so sind durchaus schnelle, überschaubare und für die Praxis ausreichende Ergebnisse möglich. Ein Vorteil der UML gegenüber anderen Notationen liegt in dem Umfang und der Erweiterbarkeit der Notation, was die Verwendbarkeit für nahezu alle Anwendungsbereiche sicherstellt.
Für die Modellierung mit der UML bietet sich die Nutzung von Software-Tools an, da die Vollständigkeit und Konsistenz eines komplexen Modells von Hand nur sehr schwer zu garantieren ist.
Im Rahmen der Projektgruppe kann die UML sehr gut genutzt werden, da in einem iterativen Prozeß die

Modellierung eines allgemeinen Labores auf einem Niveau niedrigen Detailierungsgrades möglich ist. Der Detailierungsgrad kann dann bei der Modellierung eines konkreten Labores erhöht werden, so daß ein entsprechendes Software-Tool Teile der Codegenerierung erledigen kann.

Teil II

Virtual Reality

Kapitel 6

3D - Modellierung

6.1 Zusammenfassung

Autor: Wilko Heuten

Ein wichtiger Bestandteil bei der Erstellung multimedialer Produkte stellt die 3D-Modellierung dar. Die Einsatzmöglichkeiten sind weitreichend und manchmal unersetzlich. Diese Arbeit legt die wichtigsten Aspekte, die bei der Modellierung im virtuellen Raum auftreten, nieder. Grundlegende Techniken zur Erstellung von Objekten und der Navigation im 3D-Raum werden behandelt, sowie eine Liste aktueller 3D-Software vorgestellt.

6.2 Motivation

Die vorliegende Seminararbeit gibt Einblicke in den derzeitigen Stand der 3D-Modellierung. In den ersten Abschnitten werden grundlegende Einsatzmöglichkeiten von 3D-Computergrafiken behandelt und Unterschiede zum CAD (Computer Aided Design) aufgezeigt. Desweiteren wird auf allgemeine Funktionen und Techniken der Modellierung von 3D-Körpern und auch der 2D-Modellierung, unabhängig von bestimmter Software, eingegangen. Verschiedene Vorgehensweisen werden betrachtet. Im fünften Abschnitt stehen das Aussehen und die Eigenschaften der Körper im Vordergrund. Ein wichtiger Teil der 3D-Modellierung stellt die korrekte Beleuchtung – beschrieben in Abschnitt 6 – einer Szene dar. Der 7. Abschnitt behandelt Kameras und Koordinatensysteme, um zu zeigen, wie man sich im dreidimensionalen Raum bewegen und Objekte korrekt plazieren kann. In Abschnitt 8 werden die verschiedenen Methoden des Renderns beschrieben. Dabei wird aus der vektororientierten Szene eine Bitmap erstellt. Flächen werden gefüllt, Transparenzen und Brechungen berechnet. In Abschnitt 9 erfolgt eine Auflistung aktueller 3D-Grafikprogramme. Der letzte Abschnitt faßt alles noch einmal zusammen.

6.3 Einsatz von 3D-Computergrafiken

In der heutigen Zeit werden 3D-Grafiken in einem breiten Spektrum eingesetzt. Sowohl in der Unterhaltungsindustrie als auch in wissenschaftlichen Bereichen bemächtigt man sich der generativen Computergrafik. Die 3D-Modellierung beginnt mit einfachen dreidimensionalen Schriftzügen, die mittlerweile

nicht mehr aus den Werbeblöcken des Fernsehens wegzudenken sind und endet bei fotorealistischen Bildern in aufwendigen Kinoszenen. Heutzutage ist es häufig kaum noch zu erkennen, ob ein virtueller Gegenstand im Computer generiert oder durch konventionelle Techniken erzeugt wurde.

6.3.1 Vorteile der Computergrafik gegenüber realen Modellen

Nun stellt sich natürlich die Frage, warum man überhaupt Computermodelle gegenüber realen Modellen bevorzugt. Ein wichtiger Grund stellt die Flexibilitt des Endproduktes dar. Das Computermodell läßt sich beliebig drehen und skalieren. Es kann in 4D-Programmen übernommen werden, um Animationen und Filme zu erstellen. Weiterhin ist es einfach, die Grafik von einem Ort zum anderen zu bringen, in einen beliebigen Hintergrund einzupassen und mit anderen Objeken zu verschmelzen und synchronisieren. Besonders bei größeren Projekten läßt sich viel Zeit und vor allem Geld bei der Erstellung des Modells sparen. Es kann meist ohne Probleme und kostengünstig weitergegeben und reproduziert werden. In vielen Fällen bleibt kaum etwas anderes übrig, Modelle im Computer zu generieren, falls etwa Handzeichnungen qualitativ nicht ausreichen oder in absehbarer Zeit nicht fertiggestellt werden können. Ein großer Anwendungszweig stellt die Unterhaltungsindustrei dar. Als weiterer Aspekt seien hier noch Arbeitsumgebungen wie z.B. Labore und Werkstätten in Lehre und Forschung zu erwähnen. Es kann aus Kostengründen nicht jedem Schüler ein Labor zur Verfügung gestellt werden. Also bleibt nur die Möglichkeit, virtuelle Umgebungen zu schaffen, dessen Grundlage die 3D-Modellierung darstellt. Die virtuellen Umgebungen können weiterhin Umweltbelastungen, beispielsweise durch Chemikalien, vermindern.

6.3.2 Unterschiede zwischen CAD und 3D-Modellierung

Nun werden kurz die Unterschiede zwischen CAD und der 3D-Modellierung erläutert. Für eine Vertiefung dieses Themas bzw. wie man CAD und 3D-Grafik miteinander kombinieren kann, hat Joachim Oechslin [Oec98] beschrieben. Ein Hauptunterschied besteht in der Verwendung des Endergebnisses. Mit Hilfe des CADs will man eine maßstabsgetreue, unbedingt genaue Beschreibung eines Objektes gewinnen. Es spielt dabei keine Rolle, wie ,,schön" das Modell am Ende aussieht, so daß Renderingverfahren weniger zum Einsatz kommen. Bei Architekten z.B., spielt die 3D-Ansicht eine kleinere Rolle. Es geht dort eher um den exakten Aufriß, Grundriß und Seitenriß im einzelnen, um nötige Daten für die Fertigung des Objektes gewinnen zu können.

Im Gegensatz dazu spielen die Einzelansichten und Maßstabstreue bei der 3D-Modellierung meistens keine wichtige Rolle. Dennoch sollte bei der Erstellung von 3D-Modellen auf die Einhaltung des Maßstabs geachtet werden, damit beim Zusammenfügen mehrerer Modelle keine unschönen Skalierungen notwendig werden. Desweiteren spielt das Rendering eine sehr große Rolle. Den aus Vektoren gebildeten Flächen und Volumina werden damit Aussehen, Spiegelung, Transparenz, Licht- und Schatteneffekte verliehen. Aus der Vektorgrafik wird somit eine Bitmap. Das Modell dient nicht der Fertigstellung eines Produktes, sondern einfach nur der Visualisierung.

Manchmal bietet es sich an, CAD und 3D-Modellierung zu kombinieren. Möchte beispielsweise ein Architekt seinem Auftraggeber das Aussehen des zu bauenden Hauses zeigen, so wäre es vorteilhaft, wenn die CAD-Daten in angemessener Form für den Auftraggeber visualisiert werden könnten.

6.4 2D-Modellierung

Als Grundlage der 3D-Modellierung dienen häufig zweidimensionale Objekte. Der 3D-Modellierer muß sich deshalb auch mit der Darstellung und Erzeugung von Flächen und Kurven auskennen. Im folgenden wird auf einige Arbeitsweisen der 2D-Modellierung eingegangen.

Die meisten 3D-Modellierungsprogramme besitzen die Möglichkeit, auch 2D-Objekte zu erstellen. Andere erfordern den Import von 2D-Grafiken. Neben den einfachen Methoden der Generierung von 2D-Grafiken mittels Linien, Kreisen und Polygonen gibt es die wesentlich mächtigere, aber auch kompliziertere Methode, mittels Kurven 2D-Objekte zu erstellen. Es gibt verschiedene Formen von Kurven. Hier werden die am meisten verwendeten kubischen Polynomialkurven, die *Hermite-Kurve*, die *Bézierkurve* und *Splines* erläutert.

Die *Hermite-Kurve* wird über ihre zwei Endpunkte und ihren Tangentialvektoren in den Endpunkten definiert. Sowohl die Länge als auch die Richtung ist für den Kurvenverlauf entscheidend (s. Abb. 6.1 (a)).

Bei komfortableren Grafikprogrammen kann man die Endpunkte sowie die Tangentenvektoren durch ziehen mit der Maus verändern. Zwei *Hermite-Kurven* lassen sich mit einander verknüpfen. Dabei müssen die Tangentenvektoren die gleiche Richtung besitzen, nicht aber gleich groß sein.

Bei den *Bézierkurven* müssen die Tangentenvektoren nicht auf der Kurve liegen, sondern sie werden durch zwei weitere Punkte im Raum definiert (Abb. 6.1 (b)).

So entsteht eine konvexe Hülle, die die Kurve umgibt. Zieht man an den Punkten der Hülle, so verhält sich die Kurve wie eine Art Gummiband. Für die Verknüpfung zweier Kuven gilt die gleiche Regelung wie bei *Hermite-Kurven*.

Bei den *Splines* wird die Kurve durch *n* Kontrollpunkte definiert. Beim ändern einer der Kontrollpunkte ändert sich meist die gesamte Kurve und kann daher nur schwer kontrolliert werden. Abhilfe leisten die *B-Splines* (Abb. 6.1 (c)). Sie hängen nur von einigen wenigen Kontrollpunkten ab und ein Bewegen eines Kontollpunktes ändert nur einen Teil der Kurve (lokale Steuerung). Die Kontrollpunkte eines *uniformen B-Splines*, dargestellt in parametrisierter Form, haben untereinander jeweils den gleichen Abstand bezüglich des Parameters der Kurve.

Der Vorteil der *nichtuniformen B-Splines* – die Kontrollpunkte können also frei gewählt werden – besteht darin, daß man bestimmte Punkte interpolieren kann (z.B. die Endpunkte einer Kurve) ohne die ungewünschten lineare Segmente bei den *uniformen B-Splines* zu erhalten.

Eine sehr gute Gelegenheit, den Umgang mit den Kurven zu üben, bietet eine interaktive Website [adUO96]. Die mathematischen Grundlagen lassen sich in [JDF94] nachlesen.

6.5 Modellierung von Festkörpern

Die beiden häufigsten Techniken zur Erstellung von Festkörpern sind die Anwendung von booleschen Mengenoperatoren auf Primitive sowie die Verwendung von 2D-Objekten als Grundlage zur Bildung von 3D-Objekten.

Primitive sind einfachste geometrische Formen, die in den meisten 3D-Modellierungsprogrammen angeboten werden. Dazu gehören der Würfel bzw. der Quader, der Zylinder, die Pyramide, die Kugel,

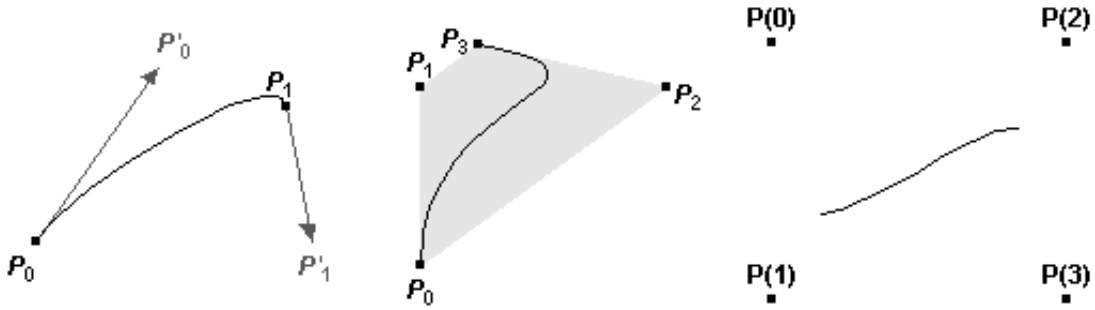


Abbildung 6.1: Kurvenarten: a) Hermite Kurve mit ihren Tangentenvektoren, b) Bezier-Kurve mit ihrer konvexen Hülle und c) uniforme B-Spline Kurve

der Kegel und der Torus. Aufgrund der einfachen Struktur sind Primitive schnell zu modellieren und zu rendern. Mittels boolescher Mengenoperatoren lassen sich viele Festkörper ,,bauen". Als boolesche Operatoren gelten der Durchschnitt und Vereinigung sowie die Differenz zweier Körper. Beispielsweise läßt sich der Grundkörper einer Mikrowelle leicht erzeugen, indem man einen kleinen Quader von einem größeren Quader abzieht und so der Hohlraum entsteht (s. Abb 6.2). Vorteil dieser Modellierungstechnik ist, daß sie oft dem realen Fertigungsprozeß entsprechen. Es lassen sich also Objekte, die durch Ausstanzen oder ,,Anschweißen" gefertigt wurden, schnell erzeugen.

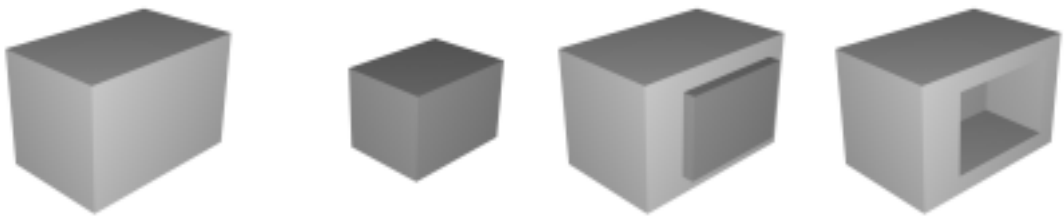


Abbildung 6.2: Boolesche Differenz zweier Quader

Eine andere Möglichkeit zur Modellierung von Körpern ist die Bildung von *Translationskörpern*. Hierzu wird ein 2D-Objekt entlang eines Pfades bewegt. Der Pfad wird meistens durch eine Kurve im dreidimensionalen Raum dargestellt. Die 2D-Objekte können wie im vorigen Abschnitt beschrieben hergestellt werden. Ist der Pfad senkrecht zur 2D-Fläche und geradlinig spricht man von *Extrusion* (s. Abb. 6.3). *Rotationskörper* lassen sich erzeugen, indem man die 2D-Fläche um eine bestimmte Achse rotieren läßt. Die Software erstellt dabei ein Flächenobjekt, was zum Nachteil hat, daß bei Nichtgefallen des entstandenen Objektes die Veränderung der Ausgangskörper keinen Einfluß auf den neuentstandenen Körper hat. Anders ist das bei den *NURBS* (Non Uniformed Rational B-Splines). Verändert man bei ihnen den oder die Ausgangssplines, so wirkt sich die Änderung auch am neuentstandenen Objekt aus. Sogenannte NURBS-Objekte sind beispielsweise die *Sweeps* oder *Loftobjekte* (s. Abb. 6.4 und 6.5)

Bei den *Sweeps* sind mindestens zwei Splines notwendig. Einer dient als Hülle, an der der andere Spline rotiert wird (s. Abb. 6.4). Unter *Lofting* versteht man die Definition eines Objektes über zwei oder mehrere 2D-Objekte, womit man z.B. die Konstruktion der Flügel eines Flugzeugs exakt nachahmen kann (Abb. 6.5).

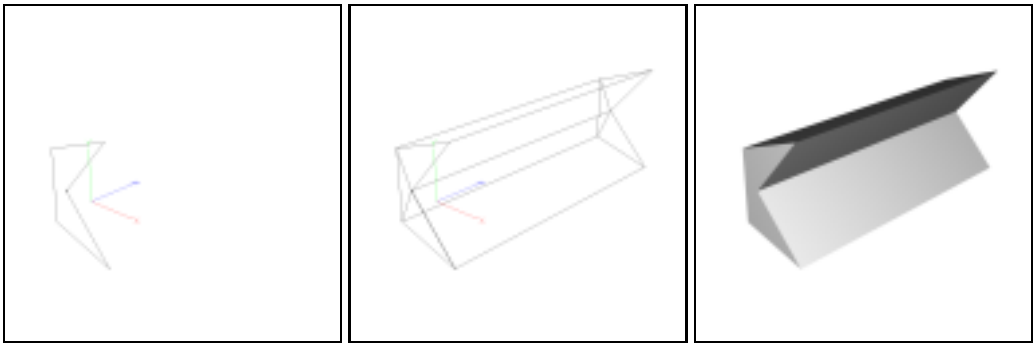


Abbildung 6.3: Extrusion

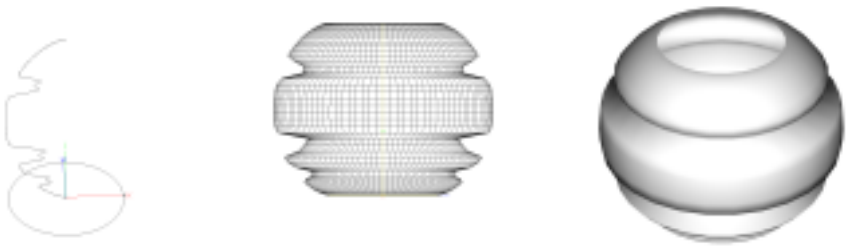


Abbildung 6.4: Sweeping

Abbildung 6.5: Lofting

NURBS haben jedoch auch einen Nachteil, wodurch die Existenz der *Translationskörper* gesichtert bleibt. Die bei der Translation neuentstandenen Objekte sind Flächenkörper, d.h. sie bestehen aus Polygonen und können mittels anderer Techniken etwa durch ,,Magneten" deformiert werden oder einer Explosion unterliegen. NURBS-Körper müssen erst durch die Software in Flächenkörper umgewandelt werden. Dies geht zwar meist zügig, jedoch ist eine Rückumwandlung der Flächenkörper zu NURBS ausgeschlossen.

Vor allem ist wichtig, darauf zu achten, welche Techniken wirklich in einem 3D-Grafikprogramm angeboten werden. Die Benennung der Technik ist im allgemeinen in verschiedenen Programmen nicht konsistent. Dazu erschweren Übersetzungen aus dem Englischen ins Deutsche und umgekehrt die Zuordnung einer Bezeichnung zu einer Technik. Die oben beschriebenen Möglichkeiten sind nicht unbedingt in jedem Grafikprogramm enthalten, manchmal werden Techniken miteinander kombiniert. Es bietet sich also an, das Benutzerhandbuch (wenn es denn eins gibt) zu Hilfe zu nehmen. Die in diesem Kapitel vorgestellen Grafiken wurden mit *Cinema 4D* unter zu Hilfenahme von [Wag94] und [cin] erstellt.

6.6 Materialien

Allein die geometrischen Ausmaße eines Körpers stellen natürlich noch keine Abbildung eines realen Objektes dar. Dazu gehören noch weitere Eigenschaften: die Oberfläche (Struktur, Farbe) und die Materialeigenschaften (Transparenz, Reflexion, Brechung, Glühen, Glanzlicht). Diese Eigenschaften haben wiederum Auswirkungen auf die Umgebung (z.B. Schattenwurf). Im folgenden werden die Verfahren beschrieben, die ein Objekt realistisch aussehen lassen sollen.

6.6.1 Farbe

Die Farbe einer Oberfläche entsteht durch die Reflexion von weißem Licht an dieser Fläche. Trifft also weißes Licht auf eine Fläche, aber nur grünes Licht wird reflektiert, so erscheint die Fläche grün. Die Farbe wird jedoch auch durch andere Eigenschaften beeinflußt. Erhöht man beispielsweise die Transparenz einer roten Fläche, wo wirkt sie immer weniger rot je höher die Transparenz ist. Für die Spezifizierung einer Farbe stehen im allgemeinen mehrere Technken zur Verfügung (RGB, HSV, CYMK . . .) und können in [JDF94] nachgelesen werden.

6.6.2 Spiegelreflexionen

Wird das Licht, das auf eine Fläche trifft, im gleichen Winkel und gleicher Intensität relativ zur Normalen der Fläche ohne Streuung reflektiert, so ergibt sich ein Spiegelbild. Objekte mit hoher Spiegelreflexion erscheinen hart und poliert. Flüssigkeiten jedoch können jedoch ebenfalls eine hohe Spiegelreflexion besitzen.

6.6.3 Glanzreflektion

Glanzreflektion entsteht ähnlich wie die Spiegelreflexion, nur wird das Licht beim Reflektieren leicht gestreut, was zu einem unscharfen ,,Fleck" führt. Diese Art der Reflexion ist verantwortlich für das *Glanzlicht* bestimmter Objekte.

6.6.4 Diffuse Reflexion

Bei der *diffusen Reflexion* wird das Licht gleichmäßig in alle Richtungen abgestrahlt. Die ist bei stumpfen Oberflächen wir z.B. Kreide der Fall.

6.6.5 Ambiente Reflexion

Ambiente Reflexion beschreibt den Grad der Reflexion des Umgebungslichts bzw. indirekten Lichts. Je höher die Reflexion, desto heller wird das ganze Objekt; Farbe und Kontraste gehen schnell verloren.

6.6.6 Transparenz

Transparenz beschreibt den Anteil des Lichts, das durchgelassen werden soll. So lassen sich Materialien wie Glas oder Flüssigkeiten modellieren.

6.6.7 Brechung

Manche Materialien brechen den Lichtstrahl, der durch transparente Objekte hindurchdringt. Das bedeutet er wird beim Eintritt und/oder beim Austritt abgelenkt. Extrem ist dieser Effekt bei Linsen, Lupen oder eines mit Wasser gefüllten Goldfischbehälters.

6.6.8 Glühen

Glühen definiert die Stärke der Lichtausstrahlung eines Objektes. Dieses Licht ist nicht mit einer Lichtquelle zu vergleichen, da es keine Auswirkungen auf ander Objekte hat. Die Intensität des Glühlichts wird zur normalen Lichtreflexion des Objektes hinzuaddiert.

6.6.9 Texturen

Da viele Objekte nicht einfarbig sind, sondern eine bestimmte Oberflächenstruktur aufweisen, gibt es die Möglichkeit Objekten eine oder mehrere Texturen zuzuweisen, um das Aussehen realistischer zu gestalten. Manchmal können sogar Videos auf Objekte gelegt werden, wodurch sich interessante Effekte ergeben. Im folgenden werden die Techniken zum Verändern der Oberfläche beschrieben:

Color Mapping: ist die einfachste Form des Texture-Mappings. Dabei wird eine 2D-Grafik über einen 3D-Körper gelegt. Es kann unter anderem benutzt werden für Gemälde an Wänden, Plazieren von Bezeichnungen auf Dosen oder Schachteln oder für das Plazieren von Skalen und Flecken.

Nahtloses Mapping: wie Color Mapping, jedoch können die Images nahtlos aneinander gereiht werden. Beispiel: Mauersteine mit Mörtel, Marmor und Holzstrukturen; alle regelmäßig wiederholten Muster.

Bump Mapping: eine Bump Map simuliert Unebenheiten auf einer Oberfläche. Sie werden normalerweise durch graustufige Bitmaps dargestellt. Je nach Programm werden dann dunkle Punkte als Vertiefungen oder als Erhöhungen dargestellt. Um dieses zu erreichen, wird je nach Grauwert die Richtung der Oberfläche geändert, so daß sich beim Rendern kleine Licht- und Schattenwechsel ergeben. Man sollte darauf achten, daß die Ecken und Kanten des Objektes jedoch immer glatt erscheinen, da die Bump Map nur ein optischer Trick ist. Die Ermittlung der sichtbaren Flächen hat schon vorher stattgefunden. Bevor ein Beleuchtungsmodell angewendet wird, werden nur die Flächennormalen der einzelnen Polygone verändert und an das Beleuchtungsmodell weitergegeben. So wird eine Orientierungsänderung simuliert.

Effect Mapping: wie Bump Maps sind auch Effect Maps meist graustufige Bitmaps. Sie können Transparenz, Glühen oder/und Reflexionen simulieren. Beispielsweise werden bei einer Glow-Map bei dunklen Pixeln die Farbe der Oberfläche dargestellt und bei helleren Pixeln die Farbe der Oberfläche aufgehellt. Diese Effect Maps sind besonders nützlich für die Simulation von Umgebungsspiegelungen und Transparenz.

Außerdem gibt es verschieden Projektionsarten, um die Textur auf das Objekt zu bringen:

Planar Mapping: ist die einfachste Art des Abbilden von Texturen auf Objekte. Man kann es sich am besten vorstellen, in dem man den Körper, beispielsweise einen Würfel durch ein Blatt mit dieser Textur drückt. Man kann es verwenden, um eine Fläche, wie z.B. eine Mauer, ohne Verzerrung mit einer Textur zu versehen.

Parametric / Proportional Mapping: die Textur paßt sich der Form des Objektes an. Wird das Objekt z.B. breiter, so wird auch die Textur gedehnt.

Cylindrical Mapping: die Textur wird um ein Objekt gelegt. Beispiel: das Label einer Cola-Dose.

Spherical Mapping: die Textur wird abgebildet wie eine Weltkarte auf einen Globus. Sie wird in Äquatornähe gedehnt und an den Polen geschrumpft.

6.7 Licht und Beleuchtung

Erst die Beleuchtung macht die Objekte in der 3D-Szene sichtbar und setzt die Stimmung einer Umgebung. Licht verändert die Kontraste und Farben der Objekte und macht Formen und Texturen deutlich. Das Setzen der perfekten Beleuchtung dauert normalerweise genauso lange oder sogar länger als das eigentliche Modellieren. Das Licht selbst wird normalerweise als nicht sichtbar dargestellt. Es verändert nur die Gegenstände, auf die es trifft. So ist es beispielsweise schwierig, Feuer zu modellieren.

Es gibt Vereinbarungen für eine ,,Standard Studio-Beleuchtung", an denen sich Photographen richten können, die aber auch beim Modellieren angewendet werden können. Die Studio-Beleuchtung besteht aus drei Lichtquellen: dem *key light*, dem *fill light* und dem *back light*. Das *key light* stellt das Hauptlicht dar. Es wird normalerweise im Vordergrund, ca. 45 Grad von der Kameraführung gedreht, aufgestellt. Das *fill light* wird verwendet, um die Schatten, die das key light wirft abzuschwächen. Standardmäßig ist es halb so stark wie das key light. Das *back light* wird benutzt, um die hinteren Kanten aufzuhellen und so den Kontrast zum dunklen Hintergrund zu verstärken. Es wird hinter das zu beleuchtende Objekt plaziert und ist genauso stark wie das key light. So ergibt sich eine Art Beleuchtungsdreieck (s. Abb. 6.6).

Abbildung 6.6: Studio Beleuchtung

6.7.1 Beleuchtungsarten

In der realen Welt gibt es verschiedene Arten von Licht. Die höher entwickelten 3D-Graphikprogramme versuchen, diese Lichtarten nachzubilden:

Punktlicht: Das Punktlicht ist das am häufigsten vorkommende künstliche Licht. Das Licht wird von einem einzigen Punkt in alle Richtungen abgestrahlt. Beispiel: Glühbirne.

Spotlicht: Das Spotlicht wird ausgerichtet wie eine Kamera. Es hat eine genaue Position im Raum und eine bestimmte Richtung. Der Winkel des Lichtkegels kann variiert werden

ambientes Licht / Umgebungslicht: *Umgebungslicht* ist besonders in natürlichen Szenen wichtig. Es ist das Licht, das von der Sonne abgestrahlt und von überall reflektiert wird. Die Lichtquelle läßt sich nicht orten.

Distanzlicht: diese Lichtquelle ist unendlich weit entfernt. Die Lichtstrahlen treffen nahezu parallel auf die Szene auf. So kann man beispielsweise das Sonnenlicht simulieren.

Außer den o.g. Lichtarten gibt es weitere Eigenschaften des Lichtes:

Farbe: farbiges Licht verändert das Aussehen der Objekte. Ein weißes Objekt, das von einem grünen Lichtkegel erfaßt wird, erscheint grün. Die Verwendung farbigen Lichtes stellt eine gute Möglichkeit dar, eine bestimmte Atmosphäre und Stimmung zu schaffen.

Intensität: beschreibt die Stärke des Lichtes. Ein Licht ohne Intensität besitzt keinerlei Auswirkung auf andere Objekte. Licht mit hoher Intensität wäscht die Objekte aus und Details gehen verloren.

Abfall: normalerweise verliert das Licht vom Zentrum des Strahls bis hin zu den Seiten an Intensität. Der Term *Abfall* beschreibt die Verringerung dieser Intensität.

Atmosphärische Abschwächung: definiert die Distanz, in der das Licht „erlischt". In der realen Welt kann dieser Wert mathematisch berechnet werden. Das Licht wird durch Partikel in der Atmosphäre absorbiert und damit abgeschwächt. Da in der 3D-Welt keine Atmosphäre modelliert wird, kann man für die *atmosphärische Abschwächung* einen Wert definieren.

6.8 Szene und Navigation

Um Objekte im dreidimensionalen Raum erstellen, bewegen, rotieren oder skalieren zu können, braucht man zwei Dinge. Zum einen muß ein Koordinatensystem definiert werden und zum anderen eine Kamera, von der aus man die Szene betrachten kann. Die meisten 3D-Grafikprogramme haben feste Voreinstellungen für Sichten von oben, von vorne und von der Seite auf die Szene und besitzen meist auch eine 3D-Ansicht (s.Abb. 6.7).

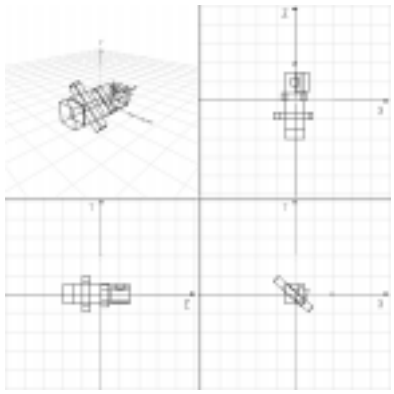


Abbildung 6.7: Die 4T-Ansicht

6.8.1 Koordinatensysteme

Für die absolute Positionierung wird ein sogenanntes *Weltkoordinatensystem* definiert. Es besteht aus drei Achsen (X,Y,Z). Der Punkt (0,0,0) wird Ursprung genannt. Die Ausrichtung der Achsen ist jedoch nicht in allen Programmen dieselbe. Beispielsweise werden manchmal die x- und z-Achse oder die negative und positive z-Achse vertauscht. Zusätzlich zum Weltkoordinatensystem besitzt jedes Objekt ein eigenes *Objektachsensystem*. So lassen sich beispielsweise Objekte leicht um die eigene Achse drehen oder relativ von ihrer Position aus bewegen. Meist läßt sich das Objektachsensystem auch verschieben, um den Rotationsangelpunkt zu verschieben. Eine Tür dreht sich um eine vertikale linke oder rechte Kante und nicht um den Mittelpunkt. Um das Drehen von Objekte zu erleichtern werden in vielen Programmen ein neues Koordinatensystem für das Objektachensystem vergeben. Dazu werden die Begriffe *pitch*, *yaw* und *roll* eingeführt. Dieses Konzept ist der Flugindustrie entnommen. Man geht immer von der Sicht des Piloten aus. Pitch bedeutet das Flugzeug bzw. ein Objekt nach oben oder unten zu schwenken (um die lokale x-Achse). Yaw ermöglicht es, das Objekt zur Seite zu drehen (um die lokale y-Achse) und roll sich über ein ,,Flügel" zu drehen (um die lokale z-Achse). Dabei spielt die Reihenfolge des Drehvorgangs keine Rolle, d.h. eine Veränderung durch *Pitch* und dann durch *Yaw* hat die gleiche Auswirkung wie eine Veränderung durch *Yaw* und danach durch *Pitch*. Die Winkel sind entkoppelt. Die Winkel werden bezüglich des Übersystems angegeben, was äußerst hilfreich bei Animationen ist. Ein Propeller dreht sich beispielsweise immer um die Achse des Flugzeugs, egal wie sich das Flugzeug im Raum bewegt. Durch die Verbindung zum Überkoordinatensystem wird jedoch ein hohes Abstraktionsvermögen verlangt und macht das *PYR-System* deswegen bei der Objektkonstruktion unbrauchbar. Siehe auch [cin]

6.8.2 Kameras

Das Konzept von Kameras hat sich beim 3D-Modellieren als erfolgreich erwiesen. Kameras besitzen eine absolute Position im Raum und haben eine bestimmte Richtung. Oft läßt sich die Brennweite einstellen und Tiefenschärfe modellieren. In einigen Programmen lassen sich mehrere Kameras aufstellen, so daß man einfach zwischen verschiedenen Ansichten hin- und herschalten kann. Kameras lassen sich auf bewegende Objekte setzen (z.B. um die Sicht aus einem fliegenden Flugzeug aufzunehmen) und manchmal lassen sich auch *Virtual Walkthroughs* kreieren. Die Umsetzung des Kamera-Konzepts sagt viel über die Mächtigkeit der 3D-Software aus.

6.9 Rendering

Die Aufgaben des Renderns bestehen in der Berechnung der sichtbaren Flächen, der Bestimmung von Lichtquellen und Lichtrichtungen, die auf die Fläche treffen, sowie der Berechnung der Reflexion des Lichts von dieser Fläche. Diese Berechnungen gehen vom Blickwinkel und der Entfernung des Betrachter aus. Als letztes kann dann die resultierende Farbe für jedes einzelne Pixel berechnet werden. Im weiteren werden die grundlegenden Methoden des Renderns dargestellt:

6.9.1 Wireframe (Drahtgitter)

Diese Art des Renderns wird meistens nur im Konstruktions-Modus verwendet. Es werden nur die Polygonnetze der Objekte aus der Sicht des Betrachters berechnet und angezeigt.

6.9.2 Hidden-line Rendering (Ermittlung sichtbarer Flächen)

Dieses Renderingverfahren gleicht dem Wireframe. Es wird jedoch berechnet, welche Flächen vom Betrachter aus sichtbar sind. Nur diese werden dann angezeigt.

6.9.3 Flat Shading

Flat Shading ist die einfachste und schnellste Art der Schattierung. Auch dieses Verfahren wird teilweise beim Konstruieren verwendet, um einen genaueren Überblick der Szene zu bekommen. Polygone und Facetten sind noch deutlich zu erkennen. Jedem Polygon einer Fläche wird nur eine einzige Farbe zugewiesen. Transparenz, Reflexion und Texturen werden nicht berechnet oder angezeigt.

6.9.4 Gouraud Shading

Das *Gouraud Shading* geht einen Schritt weiter als das Flat Shading. Die Ecken der Polygone werden zum Zentrum des Polygons hin interpoliert, was zu einer Unschärfe der Objekte führt. Es kann bei Objekten, die animiert werden und im Raum ,,fliegen” verwendet werden.

6.9.5 Phong Shading

Das *Phong Shading* Verfahren berechnet die Schattierung jedes einzelnen Pixels und legt dabei Helligkeit, Farbe und Richtung der Lichtquellen, die dieses Pixel treffen. Reflexionen dieses Lichtes von der Fläche werden jedoch nicht mehr berechnet, was bei einer Spiegelung jedoch nötig wäre. Durch Wahl geeigneter Texturen lassen sich jedoch ,,irreale" Spiegelungen und Materialeigenschaften kreieren. Dies bedeutet aber meist mehr Arbeit als beim Raytracing nötig ist. Ebenfalls nicht berechnet werden können sind Brechungen z.B. bei Glas. Vorteil dieses Renderns ist jedoch die hohe Geschwindigkeit gegenüber dem Raytracing. Es stellt die einfachste und schnellste Art des realistischen Renderns dar, solange nicht zu viele Maps (z.B. für die gefälschten Spiegelungen) benutzt werden.

6.9.6 Raytracing

Raytracing wird oft als ,,photorealistisch" bezeichnet. Es verfolgt den Pfad der Lichtstrahlen von allen Lichtquellen zur Oberfläche der Objekte bis hin zum Betrachter. Anders als beim Phong Shading werden die Strahlen jedoch auch nach dem Auftreffen auf eine Fläche weiterverfolgt und die Strahlungsintensität und -richtung neu berechnet. Dieses Verfahren ermöglicht so die Berechnung von Spiegelreflexionen, Schatten, Transparenz und Brechung. Aufgrund der komplexen Strahlenverfolgungen kann die Berechnung einer Szene *Raytracing* bis zu ca. zehn Mal solange dauern wie beim Phong Shading. Daher eignet sich manchmal das Phong Shading eher. Bei einigen Raytracing-Programmen gibt es die Möglichkeit die Rekursionstiefe der Strahlenverfolgung oder bestimmte Schwellenwerte anzugeben, um die Berechnung der Szene zu beschleunigen.

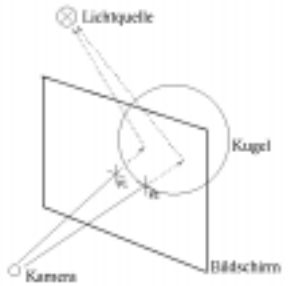


Abbildung 6.8: Funktionsweise des Raytracings

6.9.7 Radiosity

Radiosity stellt eine erweiterte Form des Raytracings dar. Algorithmen, die dieses Rendering-Verfahren beschreiben, gehen von der Erhaltung der Energie in einer geschlossenen Szene aus. Jeder von einer Fläche abgestrahlten oder reflektierten Energie entspricht der Reflexion oder Absorption durch andere Flächen. Die von einer Fläche abgegebene Energie heißt Strahlung oder *Radiosity*. Sie besteht aus der Summe der abgestrahlten Energie, der reflektierten Energie und der Energie, die durch die Fläche hindurchtritt. Es werden so also beispielsweise nicht nur Spiegelreflexionen sondern auch diffuse Reflexionen verfolgt, um noch mehr Realismus zu erzeugen. Im Gegensatz zu allen oben erwähnten Rendering-Verfahren werden alle Lichtinteraktionen eine Szene unabhängig vom Blickpunkt des Betrachters berechnet. So können relativ schnell verschiedene Darstellungen der Szene gerastert werden, da nur noch

die Ermittlung der sichtbaren Flächen und Schattierung ermittelt werden müssen. Dieses Verfahren erfordert einen noch höheren Aufwand als das Raytracing, liefert jedoch für bestimmte Szenen und Objekten mit matten Oberflächen noch realistischere Ergebnisse.

6.10 3D-Grafik Software

Nachdem die grundlegenden Konzepte der 3D-Modellierung in den vorigen Abschnitten erläutert wurden, folgt hier eine Skizzierung einiger 3D-Software-Pakete. Im folgenden werden wir sehen, daß für bestimmte Aufgaben weniger und mehr geeignetere Programme erhältlich sind. Die meisten Information sind [Lov97b] entnommen.

6.10.1 Extreme 3D 2.0 (Macromedia)

3D-Körper können als Splines dargestellt werden. Das ,,Zusammenklumpen" beliebiger Kurven oder 2D-Körper ist möglich. Mit Hilfe von Kurven lassen sich Teile der 3D-Objekte wegschneiden, sodaß sich auch mit Spline-Flächen boolesche Operatoren verwirklichen lassen. Eine Animation mit Partikeln kann realisiert werden. Environmental Maps ahmen Reflexionseigenschaften nach. Hervorragend durch sichtbare Lichtkegel. Schwächen: da das Programm kein Raytracing verwendet, können keine exakten Spiegelbilder auf Metalloberflächen und keine echten Lichtbrechungen modelliert werden.

6.10.2 Highlight Professional 2.0 (Ingenieurbüro M. Rahlff)

Eine Sammlung getrennter Programme zum Modellieren, Animieren und Rendern. Auf die Knotenpunkte eines Modells können Kopien eines anderen Modells gesetzt werden. So entsteht ein Wald von Objekten. Die Konturen einer Bitmap können extrudiert werden (*Bevels*). Objekte können ,,gesprengt" werden. Das Programm ist komfortabel und äußerst vielseitig bei der Texturgestaltung. Es gibt viele Animationsfunktionen. Schwächen: es werden nur ganzzahlige Koordinaten beherrscht mit der Folge, daß es Rundungsfehler beim Skalieren und Rotieren gibt.

6.10.3 Ray Dream Studio 4.0 (Fractal Design)

Als interne Grundlage dienen Spline-Flächen. Der Benutzer greift jedoch immer auf eine 3D-Konstruktionsskizze zu. Es gibt sehr flexible Extrusions- und Rotationsmöglichkeiten. Loft-Objekte lassen sich mit frei zeichenbaren Zwischenprofilen erstellen. Hervorzuheben ist die Inverse Kinematik-Funktion, die beim Animieren zum Einsatz kommen kann. Der Modellierungsmodus bietet Phong-Schattierung mit Texturmustern. Ergebnis: Das Progamm geht in die Breite der 3D-Modellierung. Es gibt keine wirklichen Spezialitäten.

6.10.4 Reflections 4.0 (Oberland)

Als herausragend fällt das Drag&Drop Konzept (ähnlich wie bei Cinema 4D) auf. Eigenschaften wie Texturen und Funktionen werden auf Objektnamen gezogen. Als positiv erweist sich die Glanzkurve, mit

der man Glanzlichter und Reflexstreifen für Metalle einfach erzeugen kann. Eine Relief-Funktion steht ebenfalls zur Verfügung. Animationen lassen sich bequem mittels des ,,Time-Line"-Konzeptes erstellen. Einzigartig ist die Scriptsprache, mit der sich Menüeinträge und Drag&Drop-Funktionen programmieren lassen. Schwächen: Bitmaps lassen sich nur als Farbtexturen und Bump-Maps, nicht aber als Effect-Maps zur Steuerung der Transparenz oder Spiegelung benutzen.

6.10.5 Visual Reality 2.0 (Visual Software)

Auch hier sind Modellierungsteil und Animationsteil altmodisch getrennt. Zum Funktionsumfang gehören das Biegen, Verdrillen, das Glätten (Schmelzen) mittels eines dreidimensionalen Gitters und das Morphen. Herausragend: Es können Videos als Grundlage für Farbtexturen, Bump- oder Environment-Maps benutzt werden. Größter Nachteil: es gibt wie bei Extreme 3D kein Raytracing.

6.10.6 3D Studio MAX (Autodesk)

Die Szene wird während der Konstruktionsphase schattiert dargestellt. Es nutzt die Multiprozessormöglchikeit von Windows NT aus und dank Multithreading lassen sich Körper während einer Animation noch interaktiv verändern. Alle Funktionen sind als Plug-Ins ausgeführt und können erweitert werden. Das Modellieren erfolgt prinzipiell auf Basis von Spline-Flächen und werden per Vollkörperdarstellung kontrolliert. MAX ist kein Raytracer, beherrscht jedoch einige Effekte, wie Lichtbrechung und mehrfache Spiegelungen und ist herausragend in der Erzeugung von Nebelschwaben und sichtbaren Lichtkegeln mit Schattenwurf anderer Objekte. Der Funktionsumfang enthält Inverse Kinematik, Partikelsystem (Regen, Schaum, Luftblasen), sowie Kraftfelder zum Verformen von Körpern. Animierte Objekte können durch Graviationsquellen, Deflektoren und Windstärken beeinflußt werden. Nachteil: der Preis, der weit höher ist als bei allen anderen Konkurrenzprodukten (s. auch [Sch96]).

6.10.7 Cinema 4D (Maxon)

Als herausragend gilt die ungewöhnlich übersichtliche und bedienungsfreundliche Oberfläche. Alle Einstellungen können in Voreinstellungdateien gespeichert werden, was sich gerade bei mehreren Nutzern des Systems als nützlich erweist. Weiterer Vorteil der Software ist die enorme Geschwindikeit beim Rendern gegenüber anderen Programmen, obwohl Cinema 4D nicht auf Standards wie OpenGL und damit auf Ausnutzung evtl. 3D-Grafikbeschleunigerkarten setzt. Objekte können in der Konstuktionsphase als gouraudschattierte Objekte dargestellt werden. Inverse Kinematik setzt Menschen und Tiere in Bewegung. Texturen lassen sich auf den Objekten animieren. Cinema4D bietet eine PlugIn-Schnittstelle, sodaß der Funktionsumfang erweitert werden kann. Der Funktionsumfang ist standardmäßig jedoch schon äußerst groß: Funktionen wie Zerschmelzen, Explodieren, Glüheffekte sind bereits im Lieferumfang enthalten. Schatten, Lichtkegel, Partikelsystem, Transparenz und Brechung sind kein Problem für Cinema 4D. Es erlaubt außerdem die Abspeicherung des Alpha-Kanals, was für die Nachbearbeitung wichtig ist, wenn man beispielsweise das Hintergrundbild austauschen möchte. Es gibt Exportfunktionen für QuickTime VR Movies. Die Kameras bilden Tiefenunschärfe und Lichtreflexe nach. Sie lassen sich mit Objekten bewegen und eine sogenannter ,,Virtual Walthrough", also eine Wanderung durch die Szene, läßt sich modellieren. Schwächen: ausführliche Texturdatenbank, Hilfe-Funktionen, Tutorials. (s. auch [Lov96]).

6.10.8 Ray Dream Studio 5 (Fractal Design)

Diese Software eignet sich besonders gut für das Erstellen von Comics, da es als einziges großes 3D-Programm einen Renderer enthält, der keine photorealistische Bilder erzeugt, sondern Tuschezeichnungen, Kohlestriche und abstrakte Kunstwerke. Physikalische Effekte wie Kräfte, Reibung und unelastische Stöße setzt die Software in Bewegungsabläufe um. Es stehen genügend Effektfilter für Lichtkegel, animierte Rauchschwaden, Blendenflecken und Tiefenunschärfe zur Verfügung. Der Raytracer kann weiche Schatten berechnen. Fontäne, Flammen, Wolken und Nebelschwaden lassen sich animiert darstellen. Nachteil: Der Renderer arbeitet äußerst langsam. (s. [Lov97a])

Diese Liste ist zweifelsohne nicht vollständig. Es gibt viele weitere 3D-Modellierungssoftware, die von ihrer Leistung her sicher auch erwähnt werden könnten (z.B. *Softimage*, *LightWave*). Hier soll aber nur aufgezeigt werden, daß jedes Programm besondere Stärken und Schwächen auffweist, und je nach Einsatz-Gebiet der 3D-Modellierung zum Einsatz kommen können.

6.10.9 Persistance of Vision (POVRay)

Es gibt mehrere Gründe, warum gerade dieser Raytracer gesondert erwähnt wird. Erstens ist POVRay und der Quellcode frei verfügbar, zweitens ist die Handhabung eine ganz andere als bei den o.g. Programmen und drittens liefert auch dieser Raytracer sehr brauchbare Ergebnisse.

POVRay arbeitet mit einer Szenenbeschreibungssprache. Die Beschreibung wird in einer Textdatei gespeichert und von POVray eingelesen:

```
povray +iszene.pov +oszene.tga +ft
```

Ausgabe ist ein File im TGA-Format und kann beispielsweise mit *xv* angesehen werden. *+i* legt die Eingabedatei, *+o* legt die Ausgabedatei und *+ft* legt das TGA-Ausgabeformat fest. Hier ein kleines Beispiel für eine Szenenbeschreibung (entnommen aus [Per98]):

```
#include "color.inc"

camera {
  location <0,0,-6>
  look_at  <0,0,0>
}

light_source (<-10, 10, -5> color White)

sphere {
   <0,0,0>, 2
   pigment {Yellow}
}
```

Hier eine kurze Erläuterung: `#include "color.inc"` wird nur eingeführt, damit man die Farbbezeichner *White* und *Yellow* benutzen darf. Die `camera` befindet sich an der Position `<0,0,-6>`, ist

also um -6 auf der z-Achse verschoben. Ausgericht ist die Kamera auf den Ursprung. Eine Lichquelle, die weißes Licht ausstrahlt ist um -10 auf der x-Achse, um 10 auf der y-Achse und um -5 auf der z-Achse verschoben. Auf dem Ursprung befindet sich eine Kugel mit Radius 2 (Durchmesser 4), mit einer gelben Farbtextur. Durch weiter Anweisungen im `spehre{}` Block läßt sich die Beschaffenheit der Kugel verändern. Beispielsweise würde der Zusatz `finish{phong 1}` Reflexionen auf der Oberfläche hervorrufen. Farbverläufe lassen sich ebenfalls realisieren. Außer Kugeln stehen u.a. noch Quader, Kegel, Zylinder und Tori (Reifen) zur Verfügung. Sie lassen sich alle *skalieren*, *rotieren* und *verschieben*. Die CSG-Funktionen, also das Verknüpfen von Objekten mittels *boolescher Operationen* ist genauso vorhanden wie die Möglichkeit der Benennung von Objekten. In [Per98] kann nachgelesen werden, wie ein Schraubenzieher modelliert wird. Der Code beträgt ca. 80 Zeilen.

Wie man sieht, läßt sich eine 3D-Szene auch rein textuell kreieren, wozu allerdings eine gehörige Portion Vorstellungskraft bezüglich der Raumkoordinaten notwendig ist, was einem eine kommerzielle 3D-Modellierungssoftware abnimmt. Mittlerweile bieten viele kommerziellen Programme auch den Export zum *POVRay*-Format. Offizielle Binaries und Source-Code gibt es bei [pov].

6.11 Abschliessende Bemerkungen

Die vorliegende Arbeit gibt einen Überblick über die 3D-Modellierung. Es wird das Grundlagenwissen vermittelt und darauf aufbauend Techniken zur Erstelltung von dreidimensionalen Szenen erläutert. Aus Komplexitätsgründen wurden alle Elemente, die sich mit der zeitlichen Dimension befassen, jedoch schon in vielen 3D-Grafikprogrammen vorzufinden sind, ausgelassen. Dazu gehören beispielsweise Partikelsysteme, Kamerafahrten und Animationen.

Nachdem die komplexeren dreidimensionalen Probleme wie Brechung, Transparenz und Beleuchtung nahezu gelöst wurden, liegt das Hauptaugenmerk der Entwicklung auf das User Interface Design und auf Techniken zur Berarbeitung der zeitlichen Dimension. Weiterhin wird natürlich versucht, durch verbesserte Algorithmen oder Untersützung spezieller Hardware (Multiprozessorsysteme, 3D-Beschleunigerkarten) die Geschwindigkeit des Renderns zu erhoehen.

Kapitel 7

VRML - das Web in 3D

7.1 Zusammenfassung

Autor: Christian Wichmann

In der vorliegenden Seminararbeit zum Thema *VRML* im Rahmen der Projektgruppe *Virtuelle naturwissenschaftlich-technische Labore im Internet* wird zunächst ein kurzer Überblick über die Sprache VRML und deren Geschichte gegeben. Anschließend werden die Möglichkeiten von VRML näher beschrieben, und der interne Aufbau der Sprache wird erläutert. In den nächsten Abschnitten folgen Ausführungen zum Thema VRML-Browsing und - Authoring, wobei einige Programme beispielhaft vorgestellt werden. Abschließend erfolgt eine fazitartige Zusammenfassung. Als grundlegende Literatur wurde [Has97] benutzt.

7.2 Einleitung

VRML ist die Abkürzung für *Virtual Reality Modeling Language*. Im Fachjargon wird das Akronym meist „vermel“ bzw. „Wörml“ ausgesprochen. Es handelt sich bei VRML um eine Beschreibungssprache für 3D-Szenarien, mit der man auch relativ komplexe Virtuelle Realitäten modellieren kann.

VRML ist zudem plattformunabhängig. Mit geeigneter Software kann man VRML-Szenarien auf einer Workstation genauso betrachten wie auf PCs oder dem Macintosh. Dies ist auch sinnvoll, denn VRML läßt sich nahtlos ins World Wide Web integrieren. Somit kann man VRML als das dreidimensionale Pendant zur „zweidimensionalen“ *Hypertext Markup Language* (HTML) auffassen.

VRML-Szenen lassen sich beispielweise in HTML-Seiten einbetten. Durch Anklicken von Objekten im 3D-Raum kann man dann in eine andere VRML-Szene oder auch auf eine neue HTML-Seite gelangen. Dies wird in Abbildung 7.1 veranschaulicht.

Um sich eine Vorstellung der Möglichkeiten von VRML machen zu können, sei hier eine kleine Beispielwelt skizziert:

Der Benutzer bewegt sich durch eine futuristische Galerie; er kann sich frei bewegen und natürlich auch die Gemälde an den Wänden ansteuern. Durch Klicken auf die Bilder gelangt er auf eine „gewöhnliche“ HTML-Seite mit weiterführenden Informationen über Kunstwerk und Künstler. Neben jedem Gemälde befindet sich ein Schalter, dessen Betätigung einen gesprochenen Info-Text startet. Dieser Text

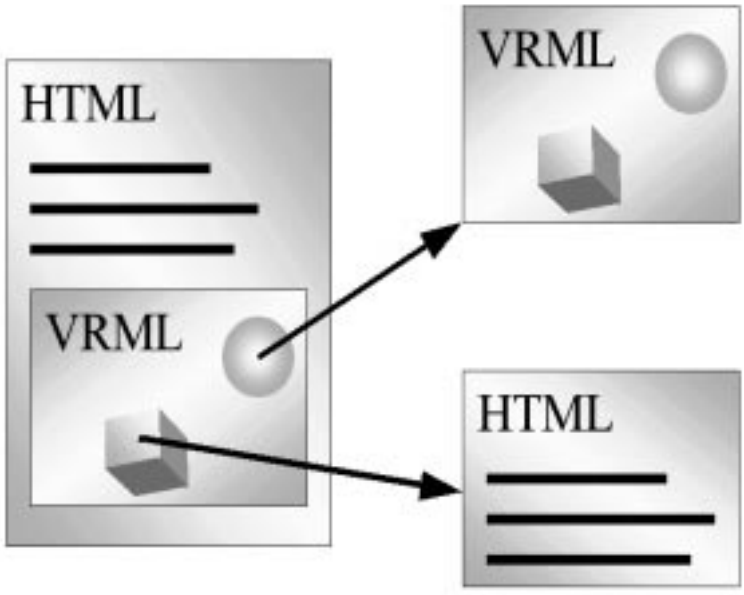


Abbildung 7.1: Verknüpfung von HTML- und VRML-Dokumenten

ist nur in der unmittelbaren Umgebung des zugehörigen Bildes zu hören. Von Zeit zu Zeit schweben Antigravitationsplattformen vorbei, auf die sich der Benutzer stellen kann, um eine Führung durch die Galerie mitzumachen.

7.3 Geschichtliches

7.3.1 Die Anfänge

Es gibt zwei Entwicklungen in der Softwarewelt, die als Wurzeln von VRML angesehen werden können:

- Die besonders seit den achtziger Jahren aufkommenden Programme für 3D-Modellierung und CAD. Beispielhaft seien hier AutoCAD (DXF), 3D Studio (3DS) und das als Freeware erhältliche Raytracing-Programm POV-Ray (POV) genannt.
- Das World Wide Web (WWW), ein Anfang der neunziger Jahre entwickelter Internet-Dienst, der es ermöglicht, hypermediale Dokumente global verfügbar zu machen. WWW-Seiten werden in der Sprache HTML erstellt und über das „Hypertext Transfer Protocol“ (HTTP) übertragen.

Als das WWW nach und nach seinen ausschließlich wissenschaftlichen Charakter verlor und zunehmend auch kommerziell genutzt wurde, wurde der Wunsch nach einer 3D-Schnittstelle für das Web laut. Zu den ersten Personen, die sich öffentlich zu diesem Thema Gedanken machten, gehörten Marc Pesce und Tony Parisi, die auch später noch eng mit der Entstehunggeschichte von VRML verbunden waren.

Im Mai 1994 fand im CERN bei Genf die erste internationale Konferenz über das WWW statt. Im Rahmen dieser Veranstaltung fand auch ein informelles Treffen von Leuten statt, die sich über das

Thema Dreidimensionalität im Internet austauschten. Hierbei wurde auch das Akronym VRML geprägt, wobei das M, in Anlehnung an HTML, zunächst noch für *Markup* stand. Später wurde es jedoch durch *Modeling* ersetzt, da die Virtuellen Welten ja tatsächlich eher „modelliert“ werden.

Kurz darauf startete das amerikanische Internet-Magazin „WIRED“ eine Mailing-Liste zum Thema VRML. So entstand ein öffentliches Diskussionsforum, in dem alle Interessierten ihre Gedanken und Vorschläge darlegen können.

Die Entwickler von VRML machten sich inzwischen auf die Suche nach einer guten Grundlage für die neue Sprache. Schließlich entschied man sich für das Dateiformat „Open Inventor“ von Silicon Graphics Inc. Dieses bildete dann auch den Kern der ersten Sprachversion.

Auf der zweiten internationalen Konferenz über das WWW stellten Tony Parisi und Gavin Bell (SGI) die Spezifikation von VRML 1.0 vor. In der Folge wurden erste VRML-Browser wie QvLib von SGI selbst und VRML-Builder entwickelt.

7.3.2 Der Weg zum ISO-Standard

Im August 1995 wurde während der SIGGRAPH 95 öffentlich zur Mitarbeit an der nächsten Version von VRML aufgerufen. Zwei Wochen nach der Messe gründete sich die „VRML Architecture Group“ , die sich vorrangig mit der Forcierung des Standardisierungsprozesses von VRML beschäftigte. Neben Marc Pesce, Tony Parisi und Gavin Bell gehörten fünf weitere technische Experten zur VAG, unter ihnen Rick Carrey.

Die VAG rief im Januar 1995 zur Abgabe von Vorschlägen für VRML 2.0 auf. Es kamen Beiträge von Apple, Microsoft, IBM Japan, der GMD und SGI. Letzterer wurde nach einer Abstimmung im Internet im März 1996 ausgewählt. Der programmatische Titel „Moving Worlds“ weist auf eine der wichtigsten Neuerungen - neben der Interaktion - gegenüber VRML 1.0 hin: die Animation von Objekten.

Im August 1996 wurde die Spezifikation von VRML 2.0 abgeschlossen und während der SIGGRAPH 96 vorgestellt. Gleichzeitig wurde die Gründung des VRML-Konsortiums, dem der Begriff VRML und die Spezifikation gehört, bekanntgegeben (siehe [VK]). Mitglieder dieses Gremiums sind Vertreter von Firmen, Forschungseinrichtungen und Universitäten, die sich u. a. um die Weiterentwicklung und Standardisierung der Sprache kümmern. Seit Februar 1998 ist auch Sun Microsystems Mitglied des VRML-Konsortiums (nach [Spe98]).

Im Dezember 1997 ist VRML 2.0 als ISO/IEC-Standard „VRML 97“ ratifiziert worden. Damit ist VRML als eine stabile Sprache offiziell anerkannt worden, die mehr darstellt als eine vorübergehende Entwicklung.

7.3.3 Blick in die Zukunft

Im VRML-Konsortium ist man der Ansicht, daß sich die Weiterentwicklung von VRML nach den Bedürfnissen des Marktes richten muß. Das zeigt sich auch in der weitestgehend öffentlich geführten Diskussion über den nächsten Sprachstandard.

Die Roadmap des Konsortiums sieht vor, die nächste Version von VRML, die „VRML 99“ heißen soll, im Frühjahr oder Sommer 1999 zu veröffentlichen. Wichtige angepeilte Verbesserungen sind unter anderem:

- Verkürzung von Ladezeiten durch Kompressions- und Streaming-Technologien
- Bessere Kompatibilität zwischen Authoring-Tools und Clients
- Einbindung von VRML in Datenbankapplikationen
- Unterstützung von Multi-User-Welten

Darüber hinaus will man Synergieeffekte mit anderen Softwareentwicklungen nutzen. Hierzu gehört die Einbindung der Java3D-API von Sun, die Verknüpfung mit dem nächsten MPEG-Standard MPEG-4 sowie das Zusammenspiel mit Microsofts „Chrome“ .

7.4 Möglichkeiten in VRML

7.4.1 Sichtbare Objekte

Die Form von geometrischen Objekten wird durch Angabe von Punkten im dreidimensionalen Raum bestimmt, die Polygonflächen bilden. Neben dem globalen Koordinatensystem besitzt jedes Objekt sein eigenes lokales Bezugssystem. Es gibt vier vordefinierte Formen:

- Quader
- Kegel
- Zylinder
- Kugel

Weitere Möglichkeiten zur Erstellung neuer Objekte umfassen:

- Angabe von Polygonmengen, die eine geometrische Form begrenzen
- Extrusion entlang einer Polylinie
- Punktmengen und Linienmengen
- Oberflächenverläufe
- zweidimensionaler Text

Das Aussehen von geometrischen Objekten wird durch Materialeigenschaften bestimmt - hierzu gehören u. a. Farbe, Glanz, Transparenz und Reflexion. Alternativ kann die Oberfläche von Objekten auch mit Texturen versehen werden. Die entsprechenden Grafiken müssen im GIF-, JPG- oder PNG-Format vorliegen. Auch digitale Videos sind als Texturen einsetzbar. Die Texturen können zusätzlich skaliert, verschoben oder rotiert werden, um beispielsweise eventuelle „Nahtstellen“ verschwinden zu lassen.

7.4.2 Licht und Schatten

Um Objekte sichtbar werden zu lassen, werden Lichtquellen gebraucht. Hierbei unterscheidet man

- gerichtetes Licht (Sonne, Scheinwerfer; theoretisch unendlich Reichweite)
- Punktlicht (Glühbirne; hat nur endliche Reichweite)
- Spotlicht (Lichtkegel; ebenfalls endliche Reichweite, eingeschränkter Winkel)
- ambientes Licht (Hintergrundbeleuchtung; setzt sich anteilsmäßig aus den anderen Lichtarten zusammen)

Abbildung 7.2: Verschiedene Arten von Lichtquellen

Der Lichteinfall äußert sich in der Schattierung sichtbarer Objekte. Grundsätzlich wird die Helligkeit einer Polygonfläche durch den Winkel bestimmt, mit dem das Licht auftrifft. Durch Interpolation kann aber bei komplexeren Schattierungsalgorithmen ein realistischeres Aussehen bewirkt werden. Zu den verschiedenen Algorithmen gehören (in der Darstellungqualität aufsteigend) Flat Shading, Gouraud Shading, Phong Shading sowie Ray Tracing. Letzteres Verfahren ist jedoch heutzutage noch bei weitem zu rechenintensiv, als daß man es in Echtzeit einsetzen könnte. Der zu benutzende Schattierungsalgorithmus wird nicht in VRML angegeben, sondern kann meist im Browser entspechend der gegebenen Hardwaremöglichkeiten gewählt werden.

Als zusätzlichen Effekt kann man „atmosphärische Abschwächung“ einsetzten. Dadurch ergibt sich eine Art Nebeleffekt, bei dem weiter entfernte Objekte schwächer zu erkennen sind.

7.4.3 Mehr reality

Um die VRML-Szene mit mehr Realismus auszustallen, kann man einen aus einem Farbverlauf bestehenden Hintergrund definieren, der aus zwei „Halbkugeln“ für Himmel und Boden besteht. Zusätzlich kann man eine Panorama-Textur wie z. B. Berge oder eine Skyline angeben.

Auch Töne kann man als Objekte in VRML-Welten einbauen. Neben gängigen Klangformaten wie Wave und MIDI kann man auch den Tonanteil digitaler Videos nutzen. Wie in der realen Welt nimmt die Lautstärke in der Entfernung immer mehr ab, und auch eine gerichtete Klangausbreitung ist realisierbar. Zusätzlich kann man 3D-Sound einsetzten, wobei die Hörerausrichtung relativ zur Klangquelle ausschlaggebend ist.

Eine der wichtigsten Neuerungen von VRML 2.0 ist die Animation von Objekten. Hierbei ist es möglich, Position und Ausrichtung sowie die Form im Sinne von Morphing kontinuierlich zu verändern. Auch Parameter wie Farbe und Reflexionsgrad lassen sich zeitabhängig kontrollieren.

Weiterhin sind mit der zweiten Sprachversion die Möglichkeit der Interaktion des Benutzers mit Objekten erweitert worden. Es gibt eine Reihe von sogenannten „Sensoren“ die bei Aktivierung z. B. Animation starten oder stoppen können. Hierzu gehören:

- Annäherungssensoren
- Berührungssensoren
- Kollisionssensoren
- Sensoren für Ziehbewegungen

Animationen selbst werden durch Zeitsensoren kontrolliert, die auch ohne Benutzeraktion aktiv werden.

Wenn die VRML-eigenen Möglichkeit zur Animation und Interaktion nicht ausreichen, ist durch die Einbindung von Skriptsprachen wie z. B. JavaScript oder Visual Basic weiterhin die Erweiterbarkeit gewährleistet.

7.4.4 Navigation

Der Benutzer kann sich in einer VRML-Szene frei bewegen, wobei Kollisionen mit Objekten jedoch durchaus möglich ist. Zum Bewegen sind verschiedene Modi wie „gehen“ oder „schweben“ wählbar, die vom Browser angeboten werden.

In einer Szene können verschiedene Ausgangspunkte, sogenannte „Viewpoints“ definiert werden, zu denen der Benutzer auch direkt springen kann.

Ein sprunghafter Standortwechsel, der als Teleportation bezeichnet wird, kann entweder zu einem anderen Viewpoint oder in eine gänzlich andere VRML-Szene führen. Alternativ kann der Benutzer so auch zu einem beliebigen anderen Internet-Dokument, beispielsweise einer WWW-Seite, „gebracht“ werden.

Der Benutzer wird in einer VRML-Welt durch einen sogenannten „Avatar“ repräsentiert, dem verschiedene Eigenschaften wie Aussehen oder Körpergröße zugewiesen werden können. Besonders interessant ist dies im Hinblick auf Multi-User-Welten, in denen mehrere Avatare aufeinander treffen und kommunizieren können.

7.5 Hinter den Kulissen

7.5.1 Aufbau von VRML-Dateien

Im folgenden soll die innere Struktur von VRML-Dateien näher erläutert werden. Trotz des recht einfachen Aufbaus kann dies aufgrund der Vielzahl verschiedener Objekte hier nur oberflächlich erfolgen.

VRML-Dateien haben die Datei-Extension .wrl bzw. .wrl.gr, falls es sich um mit GZIP komprimierte Dateien handelt, welche von vielen Browsern auch direkt gelesen werden können. Der zugehörige MIME-Typ lautet model/vrml.

Der Inhalt von VRML-Dateien ist Klartext im UTF-8-Format, von dem US-ASCII einen Spezialfall darstellt.

Durch ein einleitendes # am Zeilenanfang können erläuternde Kommentare in den Code eingebaut werden. Als Trennzeichen zwischen syntaktischen Einheiten sind verschiedene Whitespace-Zeichen wie Kommata, Leerzeichen, Tabulatoren oder Zeilenumbrüche erlaubt. Obwohl es hier keine festen Vorschriften gibt, haben sich gewisse Konventionen gebildet, wann welches Trennzeichen benutzt wird. Zwischen Groß- und Kleinschreibung wird in VRML grundsätzlich unterschieden.

Wichtigste syntaktische Einheiten einer VRML-Datei sind

- Knoten
- Definitionen von Prototypen
- Routen

Schlüsselwörter in VRML, die nicht in eigenen Definitionen verwendet werden dürfen, sind: DEF, EXTERNPROTO, FALSE, IS, NULL, ROUTE, TO, TRUE, USE, eventIn, eventOut, exposedField, field.

7.5.2 Knotentypen

Alle Objekte einer VRML-Szene werden durch Knoten repräsentiert, welche in einer streng hierarchischen Struktur organisiert sind. Man unterscheidet hier zwischen Gruppenknoten, Blattknoten und untergeordneten Knoten.

7.5.2.1 Gruppenknoten

Gruppenknoten gruppieren eine beliebige Anzahl von Kindknoten, welche Blattknoten oder wiederum Gruppenknoten sein können. So kann man beispielsweise mit dem Transform-Knoten für all seine Kindknoten gleichzeitig Verschiebungen, Skalierungen und Rotationen durchführen. Mit dem LOD-Knoten kann man eine Gruppe von Objekten in verschiedenen Detailstufen (levels of detail) definieren, von denen (in Abhängigkeit von der Entfernung) jeweils eine angezeigt wird. Es gibt

- normale Gruppenknoten (Anchor, Billboard, Collision, Group, Transform)
- spezielle Gruppenknoten (Inline, LOD, Switch)

7.5.2.2 Blattknoten

Blattknoten definieren konkrete Objekte wie Körper (Shape) oder Lichtquellen und können höchstens noch untergeordnete Knoten, aber keine Gruppenknoten oder andere Blattknoten enthalten. Man unterscheidet

- allgemeine Knoten (Script, Shape, Sound, WorldInfo)
- Lichtquellen (DirectionalLight, PointLight, SpotLight)
- Sensoren (ProximitySensor, TimeSensor, TouchSensor, ...)
- Interpolatoren (ColorInterpolator, ...)
- bindbare Knoten (Background, Fog, NavigationInfo, Viewpoint)

7.5.2.3 Untergeordnete Knoten

Untergeordnete Knoten treten innerhalb von Blattknoten oder anderen untergeordneten Knoten auf und beschreiben weitere Eigenschaften von diesen. So ist das Aussehen (Appearance) beispielsweise dem Shape-Knoten untergeordnet, und dem Appearance-Knoten wiederum ist der Material-Knoten untergeordnet. Es gibt

- geometrische Knoten (Box, Sphere, Text, ...)
- geometrische Eigenschaften (Color, Coordinate, ...)
- Aussehen (Appearance, FontStyle, Material, ...)
- Tonquellen (AudioClip, ...)

7.5.3 Aufbau von Knoten

Zu jedem Knotentyp gehören eine Anzahl von Feldern, die jeweils einen Feldtyp (z. B. SFString, SFVec3f, MFFloat oder MFNode) und einen Default-Wert besitzen. Die Werte, die Felder aufnehmen, können u. a. Strings, Vektoren, Gleitkommazahlen oder Knoten sein. Einige Felder können auch mehrere Werte aufnehmen (multi-valued field). Deren Feldtyp beginnt dann immer mit MF, im Gegensatz zu SF (single-valued field). Während Feldnamen mit Kleinbuchstaben beginnen, werden Knotennamen zur Unterscheidung immer mit einem Großbuchstaben begonnen.

```
Shape{
  appearance Appearance {
    material Material {...}
    texture ImageTexture {...}
  }
  geometry IndexedFaceSet {
    coord Coordinate {...}
    coordIndex [...]
  }
}
```

Bei allen Gruppenknoten gibt es das Feld children mit dem Typ MFNode. Hierin befinden sich alle Kindknoten des Gruppenknotens.

```
Transform {
  scale 1 1 5
  children [
    Shape {
      geometry Sphere {...}
    }
    Shape {
      geometry Box {...}
    }
  ]
}
```

7.5.4 Ereignisse und Routen

Als eine spezielle Art von Feldern sind in den meisten Knoten sogenannte „Ereignisse“ enthalten. Man unterscheidet zwischen eingehenden und ausgehenden Ereignissen.

```
field position
eventIn set_position
eventOut position_changed
```

oder kürzer:

```
exposedField position
```

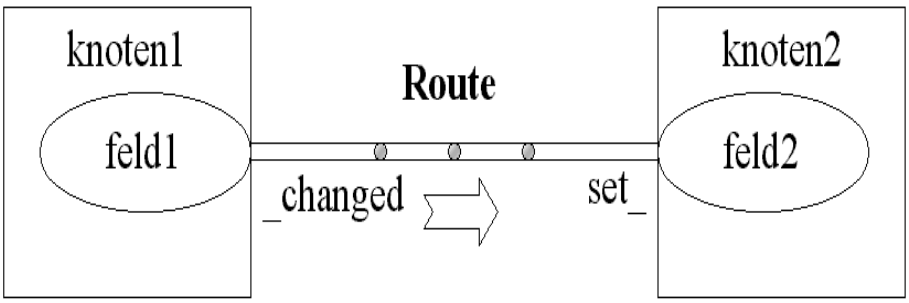
Mit Routen werden ein- und ausgehende Ereignisse zwischen Knoten verknüpft, wobei eine Kaskadierung durchaus möglich ist. Erst hierdurch wird Animation möglich: Ein TimeSensor sendet in bestimmten Intervallen Ereignisse an einen Interpolator-Knoten, und dieser generiert daraufhin neue Werte (z. B. Positionsdaten), die an ein sichtbares Objekt geschickt werden.

7.5.5 Mehrfachverwendung und Prototypen

Durch die Verwendung von DEF und USE kann ein einmal definierter Knoten mehrfach instanziert werden.

```
DEF Fussball Shape {
  geometry Sphere {radius 0.5}
}

Transform {
```




```
ROUTE knoten1.feld1_changed TO knoten2.set_feld2
```

```
  translation 3 0 -5
  children [
    USE Fussball
  ]
}
```

Auch die Erstellung von neuen, komplexen Knotentypen ist möglich. In der Definition von Prototypen schließt sich an die Felddeklaration die Beschreibung einer Mini-Szene an, die dann später wie ein neuer Knotentyp benutzt werden kann.

```
PROTO prototypname [
  eventIn      ereignistyp ereignisname_neu
  eventOut     ereignistyp ereignisname_neu
  exposedField feldtyp     feldname_neu     wert
  field        feldtyp     feldname_neu     wert
  ] {
  mini-szene (knoten, prototypen, routen)
}
```

Mit EXTERNPROTO kann auf Prototypen zugegriffen werden, die in anderen VRML- Dateien definiert worden sind.

7.5.6 Scripting

Mittels Scripting werden die Möglichkeiten komplexer Interaktionen erheblich erweitert. Der Scripting-Knoten kann

- Ereignisse empfangen, verarbeiten und versenden
- Berechnungen durchführen

- zustandsbeschreibende Werte speichern
- Werte von SF-Feldern in MF-Felder einfügen oder extrahieren
- Leistungen des VRML-Browsers nutzen (neue Knoten einfügen, neue Szenen laden, ...)
- TCP/IP-Kommunikation durchführen (z. B. für Multi-User-Welten)

Die Einbindung eines Scripts erfolgt entweder inline im url-Feld des Script- Knotens oder als Verweis auf eine externe Datei. Als mögliche Scriptsprachen kommen u. a. in Frage:

- JavaScript
- Java
- C / C++
- Perl
- TCL
- Visual Basic

7.5.7 Transformationen

Jeder VRML-Szene liegt ein dreidimensionales, rechtshändiges, cartesisches Koordinatensystem zugrunde. Das wichtigste Element zur Plazierung von sichtbaren Objekten in diesem Raum ist der Transform-Knoten. Er erlaubt

- Verschiebung (Translation)
- Rotation
- Skalierung

Bei der Transformation eines Vektors V berechnet sich der neue Vektor als V' = V * S * R * T. Da es sich bei dem Transform-Knoten um einen Gruppenknoten handelt, ist auch eine hierarchische Staffelung möglich, wobei die einzelnen Transformationen „von innen nach außen“ ausgeführt werden.

7.5.8 Eine Mini-Welt

Zum Ende dieses Abschnittes soll noch eine einfache, aber vollständige VRML- Szene vorgestellt werden. Der VRML-Quellcode sieht folgendermaßen aus:

```
#VRML V2.0 utf8
Group {
  children [
    DirectionalLight {
      direction 0 0 -1
    },
    Transform {
      translation 2 0 1
      children [
        Shape {
          geometry Sphere { radius 2.0 }
          appearance Appearance {
            material Material { diffuseColor 1 0 0 }
          }
        }
      ]
    }
  ]
}
```

Der Group-Knoten auf der obersten Ebene besitzt zwei Kindknoten: gerichtetes Licht in Richtung der negativen z-Achse und einen Transform-Knoten. Letzterer enthält als einzigen Kindknoten eine rote Kugel mit dem Radius 2, für die er eine Verschiebung um 2 Einheiten in Richtung der positiven x-Achse und um 1 Einheit in Richtung der positiven z-Achse durchführt.

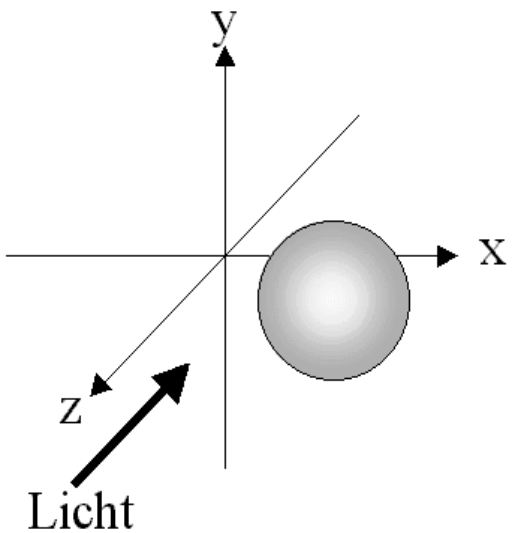


Abbildung 7.3: Lage im Koordinatensystem

7.6 VRML-Browsing

7.6.1 Varianten und Anforderungen

Die Wiedergabe von VRML-Szenen kann auf drei Arten erfolgen:

- In einem externen Viewer (z. B. Community Place von Sony); dieser wird bei Bedarf vom WWW-Browser zum Anzeigen von VRML-Szenen aufgerufen.
- In einem eigenständigen VRML-Browser (z. B. CyberGate von Blacksun, GLView von Holger Grahn); dieser ruft bei Klick auf andere Internet-Dokumente den WWW-Browser auf.
- Als integrierte Lösung in Form von Plug-ins bzw. AcitveX Controls (z. B. Cosmo Player von Cosmo Software, WorldView von Intervista); diese werden seit den 3er-Versionen vom Netscape Navigator und Microsoft Internet Explorer mitgeliefert.

Um in VRML-Welten flüssig navigieren zu können, sollte ein schneller Prozessor, eine leistungsfähige, am besten mit 3D-Funktionen ausgestattete Grafikkarte und eine durchsatzstarke Internet-Anbindung vorhanden sein.

7.6.2 Cosmo Player 2.0

Als ein Beispiel für einen integrierten VRML-Viewer sei hier der Cosmo Player von Cosmo Software ([Sof]) aufgeführt. Die VRML-Szenen können hier in HTML-Seiten mittels des EMBED-Tags eingebunden werden. Es gibt verschiedene Konfigurationsmöglichkeiten zur Anpassung an die Leistungsfähigkeit des Rechners. So können die Objekte beispielsweise nur als Wireframe dargestellt oder die Qualität der Texturen kann heruntergesetzt werden. Am unteren Fensterrand befindet sich das sogenannte „Dashboard“ mit verschiedenen Schaltflächen zur Navigation in der Szene. Über das Dashboard kann außerdem eine Hilfe im HTML-Format sowie ein „preferences“ -Dialog aufgerufen werden.

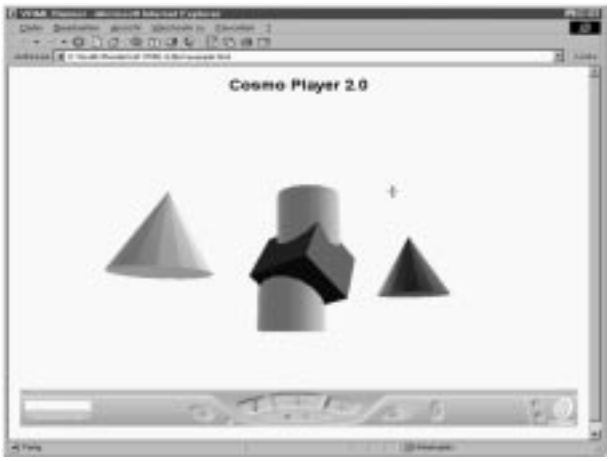


Abbildung 7.4: Cosmo Player 2.0 (Screenshot)

7.7 VRML-Authoring

7.7.1 Der Weg zur eigenen VRML-Welt

Die Erstellung von VRML-Szenen kann auf vier Arten erfolgen:

- Mit einem Texteditor. Dies ist jedoch nur für Mini-Szenen, zu didaktischen Zwecken und zur Nachbearbeitung sinnvoll.
- Mit einem VRML Builder (z. B. 3D Webmaster von Superscape, VRML Editor von Rendersoft, Community Space Conductor von Sony, VRCreator von Platinum, Cosmo Worlds von Cosmo Software). Alle führenden Produkte sind inzwischen an VRML 2.0 angepaßt.
- Mit Konvertierungstools, die Quelldateien in anderen Formaten umsetzen (z. B. vrml1to2, doomToVrml2, MAX VRML Exporter).
- Als Exportfunktion von gängigen 3D-Grafiktools wie Cinema4D und 3DMax.

Die gängige Vorgehensweise bei der Entwicklung von VRML-Szenen umfaßt folgende Schritte:

- Konzeption der Szene
- Auswahl der Tools, Einarbeitungsphase
- Grobentwurf (Ästhetik, Architektur)
- Feinschliff (Material, Texturen)
- ggf. Nachbesserungen mit einem Texteditor
- Testphase auf verschiedenen Browsern und Plattformen, Überprüfung aller Hyperlinks
- gg. Wiederholen obiger Schritte

7.7.2 Superscape 3D Webmaster

Ein bekannter VRML Builder ist 3D Webmaster von der Firma Superscape ([Sup]). Er wird mit vordefinierten Basiswelten wie Street, Garden oder Mansion ausgeliefert, die man dann beliebig erweitern kann. Viele Objekte, Sounds und Texturen kann man per Drag & Drop aus einer Bibliothek hinzufügen. Das Verschieben, Skalieren und Rotieren ist durch einen Drahtkäfig mit „Anfassern“ um das jeweilige Objekt besonders übersichtlich gelöst. Vorteilhaft ist auch, daß zusätzlich die zum Objekt gehörige „Standfläche“ angezeigt wird.

7.7.3 RenderSoft VRML Editor

Ein weiteres interessantes Tool ist der Rendersoft VRML-Editor ([Edi]). Er zeichnet sich durch besonders schnelles Rendern beim Positionieren, Skalieren und Rotieren aus. Ein „Animation Panel“ erlaubt das Erstellen von einfachen Animationen „on the fly“ . Diese Animationen bleiben natürlich auch beim Export als VRML 2.0-Datei erhalten. Negativ fällt auf, daß in der vorliegenden Version das Löschen von Lichtquellen nicht möglich ist.

Abbildung 7.5: Superscape 3D Webmaster (Screenshot)

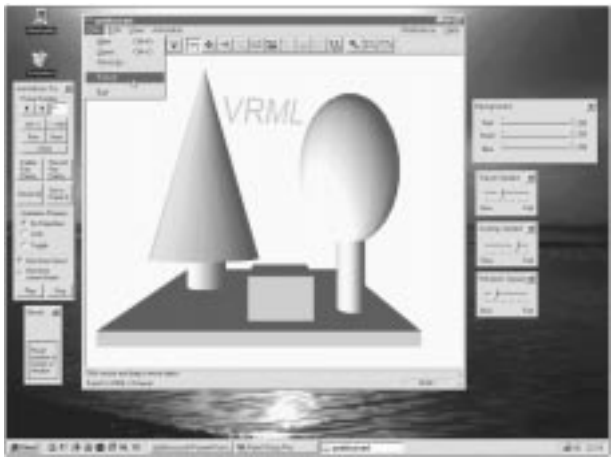


Abbildung 7.6: RenderSoft VRML Editor (Screenshot)

7.8 Abschließende Anmerkungen

Mit VRML ist das Erstellen von ansprechenden 3D-Szenarien möglich, die zahlreiche interessante Features aus dem Bereich Virtual Reality aufweisen. Die relativ nahtlose Verknüpfung mit anderen Dokumenten im Internet ist dabei ein weiterer positiver Aspekt.

Die Ratifizierung von VRML als ISO-Standard steht für die Stabilität der Sprache, und durch die Arbeit des VRML-Konsortiums, in dem viele namhafte Hersteller mitwirken, ist die zielgerichtete Weiterentwicklung und Verbreitung gesichert. Hierbei ist auch nicht zu vergessen, daß durch die öffentliche Diskussion im Internet immer wieder Anregungen von außen herangetragen werden können.

Abschließend ist zu sagen, daß die Leistungsfähigkeit von Standardhard- und -software inzwischen ausreichend ist, um eine zufriedenstellende Darstellung von VRML-Welten zu gewährleisten.

Kapitel 8

QuickTime VR und das QTVR Authoring Studio

8.1 Zusammenfassung

Autor: Jörg Appel

In dieser Ausarbeitung werden die Möglichkeiten von Apples QuickTime VR dargestellt. Insbesondere wird beschrieben, wie man eigene QuickTime VR-Medien unter Verwendung des QTVR Authoring Studios erstellen kann. Die Reihenfolge, in der die einzelnen Themen hier behandelt werden, entspricht ungefähr der Reihenfolge bei der Generierung eigener Medien, von der Aufnahme der Fotos bis zum Zusammenfügen zu einer Szene. Außerdem wird noch kurz auf die Technik eingegangen, die hinter dem Erstellen und Abspielen von QTVR-Medien steckt. Zuletzt wird auch noch auf die verschiedenen Kompressionsverfahren, die bei der Berechnung von QTVR-Filmen verwendet werden, eingegangen.

8.2 Einleitung

Bei der Entwicklung von Multi-Media-Projekten spielen Audio- und Videodaten eine große Rolle. Nachdem diese aufgenommen worden sind, müssen sie in ein Format umgewandelt werden, das auf einem digitalen Datenträger gespeichert werden kann. Dieses Format sollte zudem eine hohe Qualität bei geringer Dateigröße aufweisen. Damit nicht jeder Entwickler sein eigenes Medienformat definieren muß, haben sich einige Medienstandards durchgesetzt. Der Medienstandard von Apple ist **QuickTime**, das inzwischen in der Version 3.0 vorliegt. Es ist sowohl für Mac OS als auch für Windows erhältlich, wobei unwichtig ist, auf welcher Plattform ein QuickTime-Medium erstellt wurde. Es ist auch unter anderen Betriebssystemen abspielbar.

QuickTime 3.0 enthält Komponenten für:

- **Video/Audio**
- **MPEG**
- **MIDI**

- **3D**
- **VR**

Die Komponente **QuickTime VR** erweitert die QuickTime-Filme sozusagen um Interaktivität. Man kann einen Raum z.B. durch eine Serie von Einzelbildern oder einen Film, der einen Kameraschwenk durch diesen Raum zeigt, dem Anwender präsentieren. Genauso könnte man einen Gegenstand mittels einer Kamerafahrt um diesen herum zeigen. Doch es wäre für den Anwender schöner, wenn er sich selbst innerhalb des Raumes drehen oder den Gegenstand selbständig hin- und herdrehen könnte. Genau dies ist mit QuickTime VR möglich.

8.3 QuickTime VR

Es gibt grundsätzlich zwei verschiedene Typen von QuickTime VR-Filmen:

- **Panoramen** und
- **Objekte**.

Ein **Panorama** stellt normalerweise eine 360 Grad-Ansicht von einem Punkt aus dar. In einigen Fällen decken die Panoramen auch nur einen geringeren Bereich, z.B. nur 180 Grad, ab. Mit Hilfe der Maus kann sich der Anwender innerhalb dieses Panoramas frei nach links und rechts drehen, oftmals kann er auch noch vertikal nach oben oder nach unten schwenken. Zusätzlich besteht die Möglichkeit, durch Tastendruck zu zoomen, wobei der gerade angezeigte Ausschnitt des Panoramas auf Pixelbasis vergrößert oder verkleinert wird.

Ein **Objekt** ist ein dreidimensionaler Gegenstand, den der Anwender per Maus drehen und sich so in den meisten Fällen von allen Seiten betrachten kann. Auch hier besteht wieder die Möglichkeit, das Objekt zu vergrößern oder zu verkleinern.

Diese beiden Arten von QuickTime VR-Filmen lassen sich zu **Szenen** zusammenfassen, in die auch andere Medien integriert werden können. Eine Szene besteht dabei aus einem Panorama, das im Gegensatz zu einem einfachen Panorama interaktive Stellen, sogenannte ’hot spots’, beinhaltet, die mit anderen Medien verknüpft sind. Diese Medien werden auch Knoten (’nodes’) genannt, weshalb für diese Art von Szenen auch der Begriff ’Multi-Node-Filme’ gebräuchlich ist. Wenn ein solches Panorama betrachtet wird, wechselt der Cursor seine Form, sobald er sich über einem hot spot befindet und zeigt so dem Anwender, was dieser hier tun kann. Die Möglichkeiten bei der Erstellung von Szenen sind vielfältig: so kann man von einem Panorama in ein anderes wechseln oder einen Gegenstand ’aufheben’ und sich von allen Seiten betrachten. Hot spots können außerdem mit Klängen oder Videos verknüpft sein oder sie führen zu einer World Wide Web-Seite.

8.4 Das QuickTime VR Authoring Studio

Beim Erstellen eines eigenen QTVR-Multi-Media-Projekts würde die Reihenfolge der Arbeitsschritte wie folgt aussehen:

- Planen des Grundgerüsts der Szene und der Medien, die in der Szene gebraucht werden.
- Erstellen des Ausgangsmaterials (Fotos, Videobilder oder computererzeugte Grafiken).
- Zusammenfügen der Bilder zu QuickTime-Panoramen und -Objekten mittels eines Autorenwerkzeugs.
- Verknüpfen der verschiedenen Medien zu einer Szenerie.

Dieser Ablauf ist in Abbildung 8.1 noch einmal grafisch dargestellt.

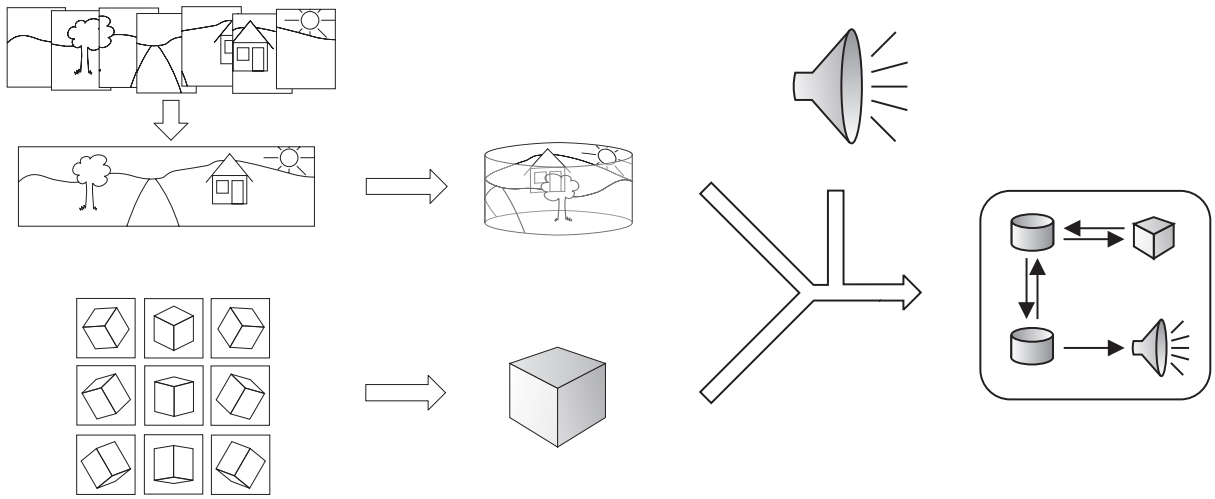


Abbildung 8.1: Ablauf beim Erstellen von QTVR-Multi-Media

Ein solches Autorenwerkzeug, mit dem QTVR-Medien erstellt werden können, ist das **QuickTime VR Authoring Studio**, das von Apple 1997 auf den Markt gebracht wurde. Dieses Tool löst die QuickTime VR 2.0 Authoring Tools Suite, die auch aus dem Hause Apple kommt, ab. Diese wiederum basiert auf dem Macintosh Programmers Workshop (MPW) und wurde hauptsächlich über eine Skript-Sprache bedient. Das QTVR Authoring Studio besitzt jetzt eine grafische Benutzungsoberfläche, wie man sie vom Mac gewohnt ist, wodurch die lange Einarbeitungszeit entfällt. Momentan ist das QTVR Authoring Studio auch nur für den Macintosh erhältlich, die erzeugten Medien sind aber natürlich auch unter Windows abspielbar. Für die verschiedene Teilschritte beim Herstellen eigener QTVR-Medien bietet das QTVR Authoring Studio jeweils ein eigenes Teilprogramm.

Im einzelnen sind dies:

- **Panorama Maker**, um aus einem einzelnen Panoramabild ein QTVR-Panorama zu erzeugen
- **Panorama Stitcher**, um aus mehreren Einzelbildern zunächst ein Panoramabild und dann ein QTVR-Panorama zu generieren
- **Object Maker**, um aus einer Serie von Ansichten eines Gegenstandes ein QTVR-Objekt zu erstellen
- **Scene Maker**, um aus verschiedenen Medien eine Szene zu erstellen

- **Project Manager**, um komplexere QTVR-Projekte zu verwalten

Der Scene Maker hebt das QTVR Authoring Studio von anderen VR-Tools ab, denn mit diesem Werkzeug ist es möglich, die oben genannten Szenen zu erstellen. Tools anderer Hersteller sind bislang nur in der Lage einzelne Panoramen und Objekte zu generieren.

Die Oberfläche der einzelnen Teilprogramme ist sehr ähnlich. Im oberen Teil des Fensters befinden sich Icons, mit denen die Einstellungen für die Filme oder Szenen vorgenommen werden, im unteren Teil werden die Grafiken angezeigt, die als Ausgangsmaterial dienen. Der obere Bereich ist in drei Teile aufgeteilt: links befinden sich Buttons, mit denen das Ausgangsmaterial betreffende Angaben gemacht werden, in der Mitte wird die Art der Ausgabe und die Dateinamen angegeben und rechts ist ein Button um die jeweilige Berechnung zu starten.

8.5 Erstellen von QTVR-Panoramen

8.5.1 Erzeugen des Ausgangsmaterials

Um eigene QuickTime VR-Panoramen herstellen zu können, braucht man zunächst das Ausgangsmaterial. Hierzu nimmt man entweder reale Umgebungen oder Gegenstände mit einer Kamera auf oder man benutzt per 3D-Software erstellte Grafik-Modelle.

Möchte man reale Umgebungen fotografieren, so ist es unwichtig, was für eine Art von Kamera benutzt wird. Normalerweise wird eine Spiegelreflexkamera mit Kleinbildfilm verwendet, man kann jedoch auch eine Video- oder eine Digitalkamera verwenden. Dabei ist allerdings zu beachten, daß die Qualität von Bildern, die aus einem Video digitalisiert oder mit einer Digitalkamera aufgenommen wurden, in der Regel schlechter ist, als die von eingescannten Fotos. Da die VR-Filme komprimiert werden sollte mit möglichst hochwertigem Ausgangsmaterial gearbeitet werden.

Um Bilder, aus denen ein Panorama generiert werden soll, aufzunehmen, muß die Kamera auf einem Stativ befestigt werden. Um einen größeren vertikalen Blickwinkel zu erreichen, sollte die Kamera im Hochformat auf dem Stativ befestigt werden. Dabei ist zu beachten, daß sich nicht die Filmebene, sondern der Brennpunkt des Objektivs über der Drehachse des Stativs befindet. Außerdem ist es wichtig, daß die Drehachse genau senkrecht zum Horizont steht. Beachtet man diese beiden Punkte nicht, so kommt es im späteren QTVR-Film eventuell zu unerwünschten Effekten, wie z.B. einem gekippten Horizont. Am besten eignen sich zur Erstellung eines Panoramas Bilder, die Gegenstände im Vorder-, Mittel- und Hintergrund haben, da das Panorama dadurch plastischer und weniger zweidimensional wirkt. Sowohl bei Innen- als auch bei Außenaufnahmen ist die Beleuchtung ein entscheidender Faktor, um gute Ergebnisse zu erhalten. Es ist nämlich sehr wichtig, daß sich die Lichtverhältnisse zwischen zwei Aufnahmen nicht drastisch ändern, da sonst im fertigen Panorama vertikale Streifen zu sehen sind. Dies ist bei Innenaufnahmen durch den Einsatz zusätzlicher Lichtquellen leicht zu vermeiden, Außenaufnahmen sollte man an einem sonnigen Tag oder bei durchgängig bedecktem Himmel machen, außerdem sollte man in beiden Fällen einen Belichtungsmesser verwenden. Bei Außenaufnahmen mit Sonnenschein muß allerdings die Linse vor direkt einstrahlendem Sonnenlicht geschützt werden, um Linsenreflexionen zu vermeiden.

Damit der Panorama Stitcher die Bilder zu einem QTVR-Panorama zusammenfügen kann, ist es wichtig, daß sich die einzelnen Bilder überlappen. Um gute Resultate zu erzielen, sollten sich die Einzelbilder um

mindestens 30 Prozent überschneiden. Da es immer passieren kann, daß ein Bild nicht richtig entwickelt wird , ist es ratsam, eine Überlappung von mindestens 50 Prozent zu wählen, um so den Verlust ausgleichen zu können. Die Anzahl der Bilder, die für ein Panorama aufgenommen werden müssen, richtet sich nach dem verwendeten Objektiv. Bei einem Weitwinkelobjektiv sind es weniger, und man erhält zusätzlich ein größeres vertikales Gesichtsfeld. Beträgt der Winkel zwischen den einzelnen Bildern z.B. 20 Grad, so ergeben sich am Ende 18 einzelne Aufnahmen.

Hat man eine Kleinbildkamera verwendet, so muß man die Fotos noch mit einem Scanner digitalisieren, bevor man sie am Computer weiterbearbeiten kann. Bei einer Digitalkamera entfällt dieser Schritt, bei Videoaufnahmen braucht man eine Grabber-Karte, um die Bilder in ein vom QTVR Authoring Studio lesbares Format zu wandeln.

Die andere Möglichkeit, Ausgangsmaterial für QTVR-Panoramen zu erzeugen, sind 3D-Render-Programme. Hier besteht der größte Teil der Arbeit darin, die dreidimensionalen Modelle der Szenen zu konstruieren. Wenn dieses getan ist, ist es relativ einfach, Bilder zu generieren, die mit dem QTVR Authoring Studio weiterverarbeitet werden können. Im Multimedia-Labor wird die Software Cinema 4D XL verwendet, die bereits von sich aus in der Lage ist, entsprechende Bilder zu generieren, man muß also keine zusätzlichen Tools benutzen. Unter den Ausgabeoptionen befindet sich eine Registerkarte 'QTVR-Panorama', auf der man den Umfang (z.B. 360 Grad) und das Format (Breite x Höhe) des Panoramas angibt. Bei der Einstellung des Formats muß man darauf achten, daß sich keine Verzerrungen ergeben, da Cinema 4D ein einzelnes Panoramabild und keine Bilderserie erstellt. Man muß also bereits vorher wissen, was für ein 'Breite zu Höhe'-Verhältnis sich bei einem kompletten Panorama ergibt, wenn man ein bestimmtes Objektiv benutzt. Dieses Verhältnis erhält man entweder durch Ausprobieren oder durch Übernehmen des Wertes eines vom Panorama Stitchers erstellten Panoramabildes. Als Dateiformat der Grafik sollte man eines der vom Authoring Studio unterstützten Formate wählen, damit man es nach der Berechnung direkt weiterverarbeiten kann, ohne es vorher konvertieren zu müssen.

8.5.2 Der Panorama Maker

Dieses Programm dient zum Erzeugen eines QTVR-Panoramas aus einem einzelnen Panoramabild, wie man es als Ausgabe von Cinema 4D XL erhält. Anhand des Panorama Makers soll hier die Bedienung des QTVR Authoring Studios, die für alle Teilprogramme ähnlich ist, kurz skizziert werden.

Zunächst muß die Grafik geladen werden, aus der das QTVR-Panorama erstellt werden soll. Dies geschieht entweder über den 'Add Image'-Button oder per 'Drag-and-drop' direkt in das Fenster. Die Grafik kann in einem beliebigen von QuickTime unterstützten Format vorliegen, z.B. als PICT, TIFF, JPEG oder GIF. Danach kann man die Grafik rotieren, falls die Ausrichtung nicht stimmt. Im nächsten Schritt legt man die Ausgabe fest, in diesem Fall, ob nur das endgültige QTVR-Panorama gespeichert werden soll, oder auch der sogenannte 'Tile-Movie', ein Zwischenschritt bei der Berechnung. Dies kann nützlich sein, da bei einer eventuellen Überarbeitung des Panoramas Zeit eingespart werden kann. Falls man die Dateien an einem anderen Ort, als dem Voreingestellten, abspeichern möchte, so kann man in den entsprechenden Feldern die Änderungen eintragen. Als nächstes klickt man den Button 'Settings', worauf sich ein neues Fenster mit vier Karteikarten öffnet. Hier nimmt man alle nötigen Einstellungen vor, die die Ausgabe betreffen.

Auf der ersten Karteikarte stellt man das gewünschte Kompressionsverfahren und die Qualität ein. Man hat die Wahl zwischen fünf verschiedenen Kompressionsmethoden, auf deren Unterschiede im Kapitel Kompression näher eingegangen wird. Die Qualität der Ausgabe läßt sich auf einer Prozentskala einstellen, wobei 100 Prozent die höchste Qualität, die mit diesem Verfahren möglich ist, (und natürlich auch

die größte Ausgabedatei) bedeutet. Auf der nächsten Karteikarte wird die Größe des Abspielfensters, der Winkel, den das Panorama abdeckt und der Ausschnitt, der beim Öffnen des VR-Films gezeigt werden soll, festgelegt. Die letzten beiden Karteikarten ermöglichen es dem Anwender, die Qualität des Films während des Abspielens (und damit auch indirekt die Abspielgeschwindigkeit) und den Grad der Verzerrungskorrektur einzustellen sowie Copyright-Vermerke einzugeben. Besonders wichtig ist hier der Menüpunkt 'Flatten To Data Fork'. Möchte man das erstellte QTVR-Panorama an andere Anwender weitergeben, so muß dieser unbedingt aktiviert werden. Insbesondere kann der spätere Film nicht auf Windows-Computern abgespielt werden. Hat man die Einstellungen abgeschlossen, klickt man den 'Make Pano'-Button, damit das QTVR-Panorama erzeugt wird. Nach Abschluß der Berechnung, öffnet sich ein MoviePlayer-Fenster, in dem man das fertige Panorama überprüfen und eventuell noch Änderungen vornehmen kann.

8.5.3 Der Panorama Stitcher

Dieses Programm erzeugt, genau wie der Panorama Maker, ein QTVR-Panorama. Als Grundlage dient kein bereits fertiges Panoramabild, sondern eine Reihe von Bildern, die jeweils einen Ausschnitt der Umgebung zeigen. Aus diesen Bildern wird zunächst ein Panoramabild zusammengesetzt, aus dem danach ein QTVR-Panorama berechnet wird.

Das Zusammenfügen des Panoramas wird vom Panorama Stitcher automatisch durchgeführt, nur in ganz speziellen Fällen könnte es nötig sein, daß der Anwender einzelne Bilder des Panoramas selbst ausrichten muß. Um ein solches Panorama erstellen zu können, ist ein kleiner Trick notwendig. Da ein Gegenstand, der auf mehreren Bildern sichtbar ist, in jedem Bild in einem anderen Winkel gezeigt wird, kann man die Bilder nicht einfach aneinandersetzen. Man würde die Grenzen der Einzelbilder deutlich sehen können, es würden sich keine fließenden Übergänge ergeben. Deshalb wird jedes Einzelbild zu einem Zylindersegment gewölbt. Dadurch ergeben sich in den Bereichen, in denen sich die Bilder überschneiden, gemeinsame Bildmuster, in denen der Panorama Stitcher nach der besten Möglichkeit der Überlagerung sucht. Das Ergebnis ist eine Panoramaansicht, in der jedoch alle geraden Linien zu Kurven geworden und die Winkel verzerrt sind.

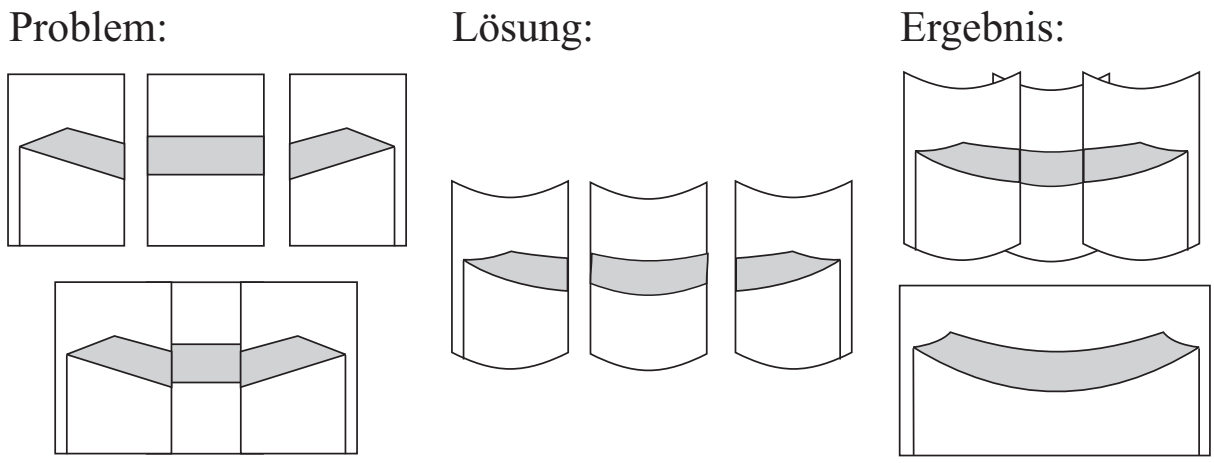


Abbildung 8.2: Zusammenfügen von Einzelbildern zu einem Panoramabild

Diese Verzerrung muß jetzt von der Abspielsoftware ausgeglichen werden, damit das Panorama perspek-

tivisch korrekt wirkt. Dazu wird in Echtzeit die Wölbung bei der Erstellung durch die gleiche Operation in die entgegengesetzte Richtung rückgängig gemacht. Dies hat zur Folge, daß die ursprünglich geraden Linien wiederhergestellt werden.

Damit das automatische Zusammenfügen des Panoramas funktioniert, müssen im Panorama Stitcher einige Parameter eingestellt werden. Zunächst einmal das bei der Aufnahme verwendete Objektiv, das aus einer Liste ausgewählt werden kann. Ist das verwendete Objektiv nicht in der Liste vorhanden, so bietet das Programm die Möglichkeit, eigene Objektive hinzuzufügen. Dies geschieht durch Eingabe der Brennweite. Ist einem diese unbekannt, so kann man diese auch durch den Panorama Stitcher anhand von zwei aneinandergrenzenden Bildern schätzen lassen. Danach muß dem Programm eingegeben werden, wie groß der Winkel zwischen den einzelnen Bildern ist. Aus diesen beiden Informationen kann der Panorama Stitcher die Überlappung der Bilder in Pixeln errechnen und somit einen Bereich ermitteln, in dem nach Übereinstimmungen gesucht wird. Falls die Bilder in der Vertikalen nicht in einer Linie aufgenommen wurden, kann man zusätzlich auch für den oberen und unteren Rand des Bildes einen Suchbereich definieren. Mit Hilfe dieser Einstellungen sollte es dem Panorama Stitcher möglich sein, das Panorama zu generieren. Sollten dennoch bei einigen Bildern die Übergänge nicht korrekt erzeugt werden, so hat der Anwender die Möglichkeit jeweils zwei Bilder manuell auszurichten.

8.6 Erstellen von QTVR-Objekten

8.6.1 Erzeugen des Ausgangsmaterials

Um Bilder für ein QuickTime VR-Objekt aufzunehmen, benötigt man entweder ein manuell oder ein motorisiert betriebenes Gestell. Möchte man ein Objekt fotografieren, das vertikal nur in einem Blickwinkel betrachtet werden soll, reicht es aus, den Gegenstand auf einem Drehteller zu plazieren und die Kamera auf ein Stativ zu stellen. Soll der Gegenstand auch aus anderen Winkeln betrachtet werden können, muß er drehbar auf einem Halter angebracht und die Kamera an einem Schwenkarm befestigt werden. Man geht bei der Aufnahme dann so vor, daß für eine Kameraposition eine komplette Objektdrehung aufgenommen wird, bevor die Kamera weiterbewegt wird. Der verwendete Objekthalter sollte möglichst klein sein, damit er im späteren VR-Objektfilm nicht auffällt. Es ist außerdem ratsam, einen neutralen Hintergrund zu benutzen, da dadurch dem Betrachter eher der Eindruck vermittelt wird, das Objekt wirklich in Händen zu halten und zu drehen anstatt drumherum zu laufen.

Ein häufiges Problem bei QTVR-Objekten ist, daß das Objekt im Film ’zittert’, also hin- und herspringt, da es in den Einzelbildern eine leicht unterschiedliche Position hat. Diese Unterschiede ergeben sich auch, wenn man sich bei den Aufnahmen die größte Mühe gegeben hat, da sie meist erst beim Digitalisieren der Fotos entstehen. Verhindern kann man dies, indem man in die Aufnahmen ein Referenzobjekt einfügt, das sich von Bild zu Bild nicht bewegt. Dann kann man nach dem Digitalisieren mittels einer geeigneten Software (z.B. Adobe After Effects) die Einzelbilder ausrichten und anschließend das Referenzobjekt mit einer Bildbearbeitungssoftware entfernen. Nicht nur aus diesem Grund hat es Vorteile, bei der Aufnahme von Objekten eine Videokamera zu benutzen, die mittels einer Grabber-Karte an den Computer angeschlossen ist. Man hat dadurch eine bessere Vorschau und kann Probleme bei der Ausleuchtung bereits vorher erkennen. Außerdem tritt das oben genannte Problem des ’Zitterns’ nicht auf, wodurch man sich sehr viel Zeit bei der Nachbearbeitung des Materials erspart. Um einen möglichst ruckfreien QTVR-Objektfilm zu erzeugen muß man nämlich sehr viele Einzelbilder aufnehmen. Wenn man das Objekt sowohl in der Horizontalen, als auch in der Vertikalen in 30 Grad-Schritten aufnimmt, ergeben sich bereits 84 Bilder, die eventuell alle nachbearbeitet werden müßten.

Auch für die Erstellung von Objektansichten läßt sich natürlich wieder eine Rendering-Software verwenden. Cinema 4D bietet wieder explizit die Möglichkeit, als Ausgabeformat 'QTVR-Object' zu wählen. Nach dem Einstellen der Winkel, in denen das QTVR-Objekt betrachtet werden soll, und der Anzahl der Bilder, erhält man als Ausgabe eine Folge von Ansichten des Objekts. Dabei ist wichtig, welche Objekte der Szene im Objekt-Manager von Cinema 4D aktiviert sind. Denn wenn sowohl die aktive Kamera, als auch das zu rendernde Objekt angewählt sind, werden auch beide nach den Vorgaben gedreht. Dies hat zur Folge, daß das Objekt immer in der gleichen Ansicht gerendert wird und sich anscheinend gar nicht dreht. Wenn die Berechnung abgeschlossen ist, können die Grafiken wieder in das Authoring Studio importiert werden.

8.6.2 Der Object Maker

Ein QuickTime VR-Objekt ist im Grunde nichts anderes, als ein navigierbarer QuickTime-Film. Denn die Einzelansichten des Gegenstandes werden als Einzelbilder eines Films abgespeichert. Dabei werden die Bilder einer Umdrehung des Gegenstandes um die vertikale Achse hintereinander gespeichert, gefolgt von der nächsten Umdrehung in einem anderen Blickwinkel. Beim Abspielen des Objekts werden die Mausbewegungen von der Abspielsoftware in Sprünge zum entsprechenden Bild des Filmes umgesetzt.

Mit dem Object Maker lassen sich aus den Einzelansichten eines Gegenstandes vier verschiedene Typen von VR-Objekten erzeugen:

- **Object**: Der Gegenstand kann vom Anwender durch Drücken der Maustaste 'gegriffen' und per Mausbewegung gedreht werden . Wenn sich der Mauszeiger am Rande des Abspielfensters befindet, wird der Film automatisch abgespielt, d.h. das Objekt dreht sich ohne daß der Anwender mit der Maus zieht.
- **Drag-only**: Verhält sich genauso wie der Typ Object, lediglich das automatische Abspielen entfällt hier.
- **Scene**: Hier kann man das Objekt nicht 'anfassen' und bewegen, sondern es wird wie mit einem Joystick gedreht. Wenn sich der Mauszeiger im rechten Bereich des Fensters befindet, verwandelt er sich in einen Pfeil nach rechts und das Objekt dreht sich nach rechts, wenn jetzt der Mausknopf gedrückt wird. Dieser Typ von QTVR-Objekt sollte vor allem dann verwendet werden, wenn der gezeigte Gegenstand real so groß ist, daß es unrealistisch wäre ihn anzufassen und zu drehen (z.B. bei einem Auto oder Flugzeug).
- **Absolute**: Bei diesem Typ kann der gezeigte Gegenstand nicht mit der Maus gedreht werden. Statt dessen werden unterschiedliche Ansichten gezeigt, je nachdem, wo im Abspielfenster der Anwender klickt.

Jeden dieser Objekttypen kann der Anwender um einen zweiten **'Viewstate'**, also einen zweiten Zustand erweitern. Dies wäre z.B. bei einem Notebook, das sonst aufgeklappt gezeigt wird, eine Ansicht in geschlossenem Zustand. Dieser zweite 'Viewstate' muß in den Winkeln der einzelnen Ansichten, sowie in deren Anzahl dem Ersten entsprechen. Beim Abspielen kann der Anwender durch Drücken des Mausknopfes zwischen den beiden Ansichten umschalten. Leider dient der Mausknopf auch zum Greifen des Gegenstandes, so daß ein Zustand nur während der Bewegung, der andere nur im Stehen zu sehen ist. Hier wäre es besser, zwischen den Ansichten durch Tastendruck umschalten zu können.

Eine weitere Möglichkeit bei der Erzeugung von Objekten ist das Einfügen einer **Animation**. Dabei werden für eine einzelne Ansicht mehrere Bilder eingefügt, die später hintereinander abgespielt werden, wenn diese Ansicht angezeigt wird. So ist es z.B. möglich, auf dem Display des Notebooks ein rotierendes Logo zu zeigen. Allerdings sollte dann für jede Ansicht, in der das Display zu sehen ist, eine entsprechende Animation eingefügt werden. Allerdings wirkt das Abspielen eines VR-Objekts mit Animation etwas ruckelig. Entweder ist die Animation abgehackt, weil der Abspieler so eingestellt ist, daß bei Mausbewegung das Objekt sofort weitergdreht wird. Oder die Bewegung 'hakt', da eine Animation zuerst zu Ende gezeigt wird, bevor das Objekt weiterbewegt wird.

8.7 Erstellen von QTVR-Szenen

8.7.1 Der Scene Maker

Eine QTVR-Szene besteht aus einer Menge von Knoten ('nodes'), wie z.B. Panoramen, Objekten oder anderen Medien. Diese Knoten werden im Scene Maker auf einer sogenannten 'map', quasi einem Grundriß der Szene, gesetzt. Hierbei kann man vier verschiedene Typen von Knoten setzen, jeweils einen für

- **QTVR-Panoramen**
- **QTVR-Objekte**
- **URLs** (also Webseiten)
- **Blobs** (ein Sammelbegriff für alle übrigen Arten von Medien)

Diese Knoten können dann untereinander durch **'links'** verknüpft werden. Bei Panoramen und Objekten können diese Verknüpfungen in beide Richtungen führen, so daß man auch wieder zurückkehren kann. Bei URLs und Blobs ist dies nicht möglich.

Die Navigation innerhalb einer QTVR-Szene erfolgt dann über sogenannte **'hot spots'**, die in die Panoramen der Szene eingefügt werden. Die 'hot spots' bezeichnen Flächen innerhalb der Panoramen, die bei Klicken mit der Maus eine Verknüpfung aktivieren. Wenn man einen 'hot spot' gesetzt hat, muß man anschließend noch einstellen, in welchem Blickwinkel das Ziel der Verknüpfung zu sehen sein soll. Dabei sollte möglichst die 'Bewegungsrichtung' des Betrachters erhalten bleiben, damit dieser innerhalb der Szene nicht die Orientierung verliert. Bei einer Verknüpfung zu einer World Wide Web-Seite, wird beim Anklicken der externe Browser zum Anschauen dieser Seite geöffnet.

Möchte man in seine Szene auch Verknüpfungen zu anderen Medien, also zu 'blobs' einfügen, so kann man mit dem QTVR Authoring Studio nur den 'hot spot' erstellen. Man kann nicht genau angeben, was für eine Aktion ausgelöst werden soll. Dazu braucht man ein weiteres Autorenwerkzeug, wie z.B. Macromedia Director. Mit diesem Programm würde man die Szene, in die man zuvor eine Verknüpfung mit einem 'blob' eingefügt hat, importieren. Jeder 'hot spot' innerhalb der Szene hat eine ID, über die man jetzt mit Director die gewünschte Aktion zuweisen kann, z.B. das Abspielen eines Sounds.

8.7.2 Größere QTVR-Projekte verwalten

Für die Erstellung größerer Projekte, wie z.B. komplexere Szenen, stellt das QTVR Authoring Studio ein weiteres Tool zur Verfügung, den **Project Manager**. Er hilft dabei, die Medien, die zu dem Projekt gehören, zu verwalten. Er ist vor allem dann nützlich, wenn sich das Ausgangsmaterial während der Erstellung noch ändert, bzw. überarbeitet wird. Sämtliche Einstellungen für die einzelnen Medien des Projekts werden vom Project Manager gesichert. Bei einer Aktualisierung werden dann automatisch nur die QTVR-Medien neu berechnet, deren Ausgangsmaterial sich geändert hat. Außerdem bietet der Project Manager die Möglichkeit, mehrere Versionen eines Projekts zu verwalten, z.B. eine hoch- und eine niedrigauflösende. Eine hochauflösende Version zur Erstellung einer CD, die niedrigauflösende zur Verbreitung übers Internet.

8.8 Kompression

Die Kompression reduziert die Dateigröße der Filme - je nach Verfahren - unterschiedlich stark.

Für die Erzeugung von QTVR-Medien sind folgende Kompressionsmethoden am besten geeignet:

- **Animation**
- **Cinepak**
- **Graphics**
- **Photo JPEG**
- **Video**

Das gewählte Kompressionsverfahren bei der Berechnung von QTVR-Medien ist deshalb so wichtig, da es über die Bildqualität der Ausgabe entscheidet. Jedes Verfahren ist auf einen bestimmten Typ von Ausgangsmaterial spezialisiert, z.B. auf Fotos oder Video, so daß man ungefähr abschätzen kann, welches Verfahren zu benutzen ist. Außer Animation bei 100 Prozent verschlechtern alle Kompressionsmethoden die Bildqualität, ja nach Verfahren und Rate unterschiedlich stark. Zusätzlich weisen die Kompressionsverfahren unterschiedliche Kompressions- und Dekompressionszeiten auf. Die Zeit, die zur Kompression benötigt wird ist nur bei der Erstellung der Medien wichtig, die Dekompressionszeit bestimmt hingegen, wie schnell ein QTVR-Film abgespielt werden kann. Das bedeutet, daß man zur Vorschau ein Verfahren benutzt, das schnell komprimiert, bei der Erstellung der endgültigen Version aber ein Verfahren verwendet, das langsamer komprimiert, aber eine hohe Qualität beim Abspielen sichert.

8.9 Fazit

Mit QuickTime VR kann man recht einfach interaktive Ansichten von realen Umgebungen und Gegenständen erstellen und wiedergeben. Jedoch ist der Begriff ’Virtual Reality’ (VR) etwas übertrieben, da der Anwender in einer Szene nur von einem Punkt zum nächsten Springen kann, sich aber nicht innerhalb eines Ortes frei bewegen kann. Schade ist auch, daß die Panoramen nicht mit Umgebungsgeräuschen versehen werden können, die beim Drehen aus unterschiedlichen Richtungen kommen.

Komp.-verfahren	ideales Ausgangs-material	Komp.-geschwin-digkeit	Dekomp.-geschwin-digkeit	Komp.-rate	Ausgabe-qualität	Anmerkungen
Animation	computer-generierte Animationen	+	+	--	++	schlechte Kompressions-rate
Cinepak	Video	-	+	+	-	Ausgangsmaterial sollte hohe Qualität haben
Graphics	computer-generierte	o	+	-	o	für 8 bit Grafiken am besten geeignet
Photo JPEG	Animationen Fotos oder Video	o	-	++	+	kann dazu führen, daß Filme "ruckeln"
Video	Video	+	+	+	o	am besten als Vorschau bei der Bearbeitung geeignet

Abbildung 8.3: Kompressionsverfahren im Überblick

Das QTVR Authoring Studio erleichtert die Erstellung eigener QuickTime-Medien ungemein. Durch die gute Benutzerführung erzielt man schnell die gewünschten Resultate. Allerdings wäre es schön, wenn die Einbindung anderer Medien vollständig innerhalb des Authoring Studios möglich wäre, ohne ein weiteres Autorenwerkzeug benutzen zu müssen.

Teil III

Elektronische Lehre

Kapitel 9

Rechnergestützte Kooperation

9.1 Zusammenfassung

Autor: Holger Schneider

Im Rahmen dieses Berichts wird untersucht, in wieweit rechnergestützte Kooperation im Verlauf der Projektgruppe ‘Virtuelle naturwissenschaftliche Laboratorien im Internet‘ eine Rolle spielen könnte. Ausgehend von möglichen Berührungspunkten der künftigen Projektgruppenarbeit mit diesem Thema wird zunächst der Begriff der rechnergestützten Kooperation an einem durchgehenden Beispiel erläutert und anhand des in [Bur94] eingeführten Modells präzisiert. Später werden Aspekte kooperationsunterstützender Werkzeuge beleuchtet, um schließlich Anforderungen an ein universell einsetzbares Werkzeug zur Kooperationsunterstützung vorstellen zu können. Abschließend wird in einem Fazit kurz erörtert, inwieweit das in diesem Bericht vermittelte Wissen für die weitere Arbeit der Projektgruppe genutzt werden kann.

9.2 Rechnergestützte Kooperation im Kontext der Projektgruppe

Ein wesentliches Ziel unserer Projektgruppe wird es sein, verschiedene Laborumgebungen auf ihre Gemeinsamkeiten hin zu untersuchen, um ein umfassendes Modell für Laboratorien bilden zu können. Haben wir ein allgemeines Labormodell aufgestellt, so möchten wir, auf diesem Modell basierende Werkzeuge zur Verfügung stellen, die Entwickler bei der Realisierung virtueller Labore unterstützen.
Ein Berührungspunkt mit der Thematik der rechnergestützten Kooperation ergibt sich, wenn wir in unserem allgemeinen Labormodell die Zusammenarbeit von Menschen berücksichtigen wollen. In diesem Moment benötigen wir eine genaue Vorstellung davon, was man im Detail unter rechnergestützter Kooperation versteht. Abschnitt 2 dieses Berichts stellt deshalb ein Modell für rechnergestützte Kooperation vor.
Ein weiterer Berührungspunkt ergibt sich, wenn wir ein Werkzeug erstellen wollen, das das kooperative Arbeiten von Menschen in virtuellen Laboratorien unterstützt. Dann müssen wir wissen, welche Anforderungen man an ein kooperationsunterstüzendes Werkzeug stellen kann. In den Abschnitten 3 und 4 werden deshalb Anforderungen an Kooperationswerkzeuge vorgestellt.

9.3 Universelles Modell rechnergestützter Kooperation

Um ein Werkzeug für das kooperative Arbeiten von Menschen in virtuellen Laboratorien entwerfen zu können, muß man zunächst einmal wissen, was 'rechnergestützte Kooperation' überhaupt bedeutet. In der Arbeit von [Bur94] wird ein universelles Modell rechnergestützter Kooperation definiert. Dieses Modell soll in diesem Abschnitt vorgestellt werden. Es erfaßt sämtliche Eigenschaften und Zusammenhänge, um beliebige Kooperationen beschreiben zu können. Ein solches Modell ist deshalb insbesondere auf jede Groupware-Anwendung anwendbar. Da unsere virtuellen Laboratorien Groupware-Anwendungen darstellen, können wir uns darauf verlassen, daß auch alle für uns wesentlichen Eigenschaften und Zusammenhänge in diesem Modell verankert sind.
Bevor dieses Modell vorgestellt wird, soll ein Beispielszenario für eine rechnergestützte Kooperation angegeben werden. Auf dieses Beispiel wird sich dann bei der Erklärung der Modellelemente bezogen, um sie leichter verständlich zu machen.

9.3.1 Beispielszenario einer rechnergestützter Kooperation

In dem Projekt 'Virtuelles Wochenmagazin' haben sich Menschen zusammengefunden, um ein wöchentlich im Internet erscheinendes Magazin zu gestalten. Das gesamte Team umfaßt zur Zeit 20 Mitarbeiter, die über ganz Deutschland verteilt sind und gewöhnlich nur über das Internet miteinander in Verbindung treten. Das Team hat sich seit seiner Entstehung in mehrere Abteilungen untergliedert, von denen die folgenden drei betrachtet werden sollen:

- Text
- Bild
- Montage

Die Menschen, die in der Abteilung 'Text' tätig sind, verfassen die Artikeltexte. Die Illustration der Artikel mit Bildmaterial geschieht in der Abteilung 'Bild'. Letztlich wird die Zeitung in der Abteilung 'Montage' aus allen Einzelteilen, wie Texten, Bildern, Grafiken, Anzeigen, Werbungen, usw. zusammengesetzt und im World Wide Web angeboten.
In der nächsten Ausgabe des Virtuellen Wochenmagazins, die am 15.5.98 herausgegeben wird, soll ein Artikel über den amerikanischen Präsidenten Bill Clinton inklusive zweier Fotos als Leitartikel im Magazin erscheinen.

9.3.2 Die Elemente des Modells

Das Modell zur rechnergestützten Kooperation besteht aus einem individuellen und einem kooperativen Teil. Im individuellen Teil des Modells wird beschrieben, welche Merkmale und Zusammenhänge für die Beschreibung des Aufgabenbewältigungsprozesses eines Menschen relevant sind. Dieser Teil des Modells modelliert somit die Arbeit jedes einzelnen, an der Kooperation beteiligten Menschen. Die Elemente des individuellen Teils des Modells sind: Subjekt, Rolle, Rolleninstanz, Objekt, Aktivität und Ziel.
Der kooperative Teil des Modells beschreibt diejenigen Merkmale und Zusammenhänge, die relevant

werden, wenn mehrere Menschen zielgerichtet zusammenarbeiten wollen. Er modelliert also im wesentlichen, wie sich die an der Kooperation beteiligten Menschen untereinander abstimmen müssen, um das gemeinsame Ziel zu erreichen. Die Elemente des kooperativen Teils des Modells sind: Kommunikation, Koordination, Kooperation und Kooperationskontext.
Im Folgenden sollen zuerst die individuellen und dann die kooperativen Elemente des Modells erläutert werden.

9.3.2.1 Individuelle Modellelemente

SUBJEKT ist eine Person oder eine Gruppe von Personen. Subjekte verfügen über Ziele und haben unterschiedliche Rollen im Kontext der rechnergestützten Kooperation. Subjekte sind Eigentümer oder Mitbenutzer von Objekten.

Jeder einzelne Mensch, der an der Kooperation ‘Virtuelles Wochenmagazin‘ beteiligt ist, ist nach dieser Definition ein Subjekt. Alle Menschen, die in einer der drei betrachteten Abteilungen zusammenarbeiten, werden ebenfalls als Subjekte verstanden. Die Ziele der Abteilungen liegen klar auf der Hand: Sie möchten je nach ihrer Funktion entweder eine Menge von Artikeln erstellen, eine Menge von Bildern und Grafiken erzeugen, die die einzelnen Artikel illustrieren sollen oder alle Einzelteile im Magazin zusammenfügen. Bei ihrer Arbeit erstellt eine Mitarbeiterin der Abteilung ‘Text‘ ein Objekt des Typs ‘Formatierter Text‘ und ist somit Eigentümerin des Objekts. Ein anderer Mitarbeiter der Abteilung muß auf den erstellten Text zugreifen können, um ihn Korrektur zu lesen. Er ist Mitbenutzer des Objekts.

OBJEKT ist einem oder mehreren Subjekten zugänglich. Objekte können temporär oder permanent existieren. Sie haben einen Zustand, der durch Aktivitäten manipuliert werden kann (Erzeugen, Vernichten, Zustand ändern).

Nach dieser Definition ist jeder in der Kooperation ‘Virtuelles Wochenmagazin‘ auftauchende Arbeitsgegenstand ein Objekt. Insbesondere ist eine email ein Beispiel für ein temporäres Objekt (sie wird meist nach Erhalt und Lesen gelöscht) und Fotos bzw. Artikel Beispiele für permanente Objekte (sie sind wesentliche Bestandteile des Magazins und werden archiviert).

ROLLE stellt einen Platzhalter für Aufgaben, Rechte, Pflichten usw. eines Subjekts in einer Kooperation dar. Dieser ist je nach Anwendung auszufüllen.

Beispiele für Rollen innerhalb der Kooperation ‘Virtuelles Wochenmagazin‘ sind Bildgestalter, Texter und Layouter. Als Rollenmerkmale könnte man definieren: Mitarbeiter der Abteilung X, Teamsprecher der Abteilung X, darf Objekt Y lesen, darf Objekt Y schreiben, muß an der Hauptredaktionssitzung teilnehmen.

ROLLENINSTANZ wird aus einer Rolle erzeugt, indem für jedes Rollenmarkmal der Rolle eine Merkmalsausprägung ausgewählt und dann einem oder mehreren Subjekten zugewiesen wird. Auf diese Weise erhalten Subjekte genau definierte Eigenschaften.

Mitarbeiter 5 bekommt eine Rolleninstanz der Rolle Bildgestalter mit folgenden Merkmalsausprägungen zugewiesen: Angehöriger der Abteilung Bild, kein Teamsprecher, darf Objekt ‘Foto von Clinton‘ lesen, darf ‘Foto von Clinton‘ schreiben, muß nicht an der Hauptredaktionssitzung teilnehmen.

AKTIVITÄT wird durch ein Subjekt initiiert. Sie operiert auf Objekten und kann deren Zustand ändern. Aktivitäten können aus Einzelaktivitäten zusammengesetzt sein.

Mitarbeiter 2 in seiner Rolle als Texter erzeugt anfangs ein leeres Textobjekt und verändert solange den Zustand des Objekts (durch Aneinanderreihung von Wörtern), bis der erstellte Text seinen Ansprüchen genügt. Die Aktivität besteht dabei in der Erstellung des Textes. Jeder Edititiervorgang kann als Einzelaktivität angesehen werden.

ZIEL ist ein erwünschter Objektzustand, d.h. der (angestrebte) Endzustand eines Objektes, der sich aus dem Anfangszustand des Objekts durch Anwendung einer Folge zustandsmanipulierender Aktivitäten ergibt. Ein Ziel umfaßt zusätzlich noch Regeln zur Bewertung des Zielerreichungsgrads. Wichtige Zieltypen sind Gesamt- und Teilziele, wobei sich das Gesamtziel durch die Erreichung von Teilzielen ergibt.

Das Gesamtziel der Kooperation ‘Erstellung eines Clinton-Artikels‘ ist definiert durch den erwün-schten Objekzustand ‘druckreifer Artikel‘. Um diesen angestrebten Objektzustand zu erreichen, muß der Text des Artikels erstellt und korrigiert sein (ein Teilziel), es müssen zwei Fotos von Clinton vorliegen (ein weiteres Teilziel) und Text und Fotos müssen zu einem Artikel zusammengefügt werden (ein drittes Teilziel), der erst nach dem Fein-Layout als druckreif gilt. Die Aktivitätsfolge, die nötig ist um das Teilziel ‘zwei Fotos von Clinton‘ zu erreichen, könnte aus folgenden Einzelaktivitäten bestehen: Auswahl geeigneter Fotos, Besprechung mit dem Texter (passen die ausgewählten Fotos zum Text), Nachbearbeitung der Fotos (bezüglich Größe, Farbe, usw.).

9.3.2.2 Kooperative Modellelemente

KOMMUNIKATION schafft Verbindung zwischen Subjekten. Wie in Abbildung 1 dargestellt, kann eine Kommunikation allgemein als die Manipulation eines Objektzustands seitens des Senders und die Wahrnehmung des Objektzustands seitens der Empfänger verstanden werden. Dabei gewinnen die Empfänger die vom Sender angestrebte Information aus dem Zustand des Kommunikationsobjekts. Hinsichtlich der Adressierungsform einer Kommunikation ist zwischen direkter und anonymer Adressierung zu unterscheiden. Direkte Adressierung bedeutet, daß der Sender mit einer genau definierten Menge von Empfängern kommuniziert, während bei der anonymen Adressierung dem Sender nicht bekannt ist, wer zu den Empfängern seiner Nachricht gehört.

Ein Beispiel für eine direkte Adressierung ist eine email, die Mitarbeiter 2 der Abteilung Montage an Mitarbeiter 5 der Abteilung Bild verschickt, um ihm mitzuteilen, daß eines der von ihm eingereichten Fotos von Clinton zu klein ist. Ein Beispiel für anonyme Adressierung ist ein Textobjekt ’allgemeine Ankündigungen’, das mit allgemeinem Lesezugriff in der virtuellen Redaktion abgelegt (’ans schwarze Brett geheftet’) wird.

KOORDINATION benötigt in jedem Falle Kommunikation und dient der Abstimmung von Teilzielen, Rollen und Zeitvorgaben unter den an einer Kooperation beteiligten Subjekten. Koordination beinhaltet

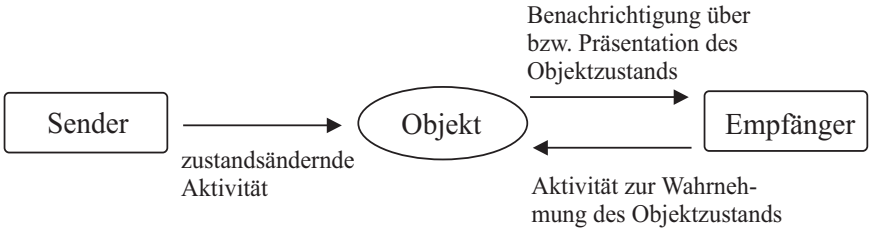


Abbildung 9.1: Kommunikationsmechanismus

zusätzlich die Beobachtung und Überwachung von Aufgabenbearbeitungen. Hinsichtlich der Koordinationsstruktur unterscheidet man zwischen hierarchischen und lateralen Ausprägungen. In hierarchischen Strukturen koordiniert ein Subjekt eine Menge von untergeordneten Subjekten, indem es durch Befehle z.B. Teilziele für seine Untergebenen definiert. In lateralen Strukturen koordinieren sich die beteiligten Subjekte selbst. Sie sind dabei selbst für die Definition, Abstimmung und Verteilung von Teilzielen innerhalb des Teams verantwortlich.

Für die ‘Virtuelle Wochenschau‘ ist Koordination nötig, um z.B. die Teilziele ‘Zwei Clinton-Fotos‘ und ‘Erstellung des Artikeltextes‘ zum 13.5.98 zu formulieren und an die entsprechenden Mitarbeiter zu vergeben. Dabei können diese Teilziele, Rollen und Zeitvorgaben entweder durch Abstimmung unter den Mitarbeitern selbst (laterale Struktur) oder durch einen Koordinator (hierarchische Struktur) definiert bzw. vergeben werden.

KOOPERATION beschreibt den sich wiederholenden Prozeß der Koordination von Teilzielen und der anschließenden Erreichung dieser Teilziele durch Aktivitäten. Am Ende dieses Prozesses ist das Gesamtziel (Kooperationsziel) erreicht. Auf die Erreichung des Kooperationsziels folgt meist eine Bewertung des Erfolgs der Kooperation. Dabei soll vor allem festgestellt werden, ob und wo Arbeitsabläufe oder Koordinationsvorgänge optimiert werden können.

Das Kooperationsziel für unser Beispiel lautet: Ein Artikel über Clinton inklusive zweier Fotos soll am 15.5.98 als Leitartikel im ‘Virtuellen Wochenmagazin‘ erscheinen. Um dieses Gesamtziel zu erreichen ist z.B. die unter dem Modellelement Koordination beschriebene Abstimmung von Teilzielen mit anschließender Erreichung dieser Teilziele durch die entsprechenden Aktivitäten der Mitarbeiter nötig.

KOOPERATIONSKONTEXT wird definiert durch alle an einer Kooperation beteiligten Subjekte, Objekte und Aktivitäten.

9.4 Kooperationsunterstützende Werkzeuge

Wir haben im vorigen Abschnitt gesehen, daß rechnergestützte Kooperation auf den Ebenen der Kommunikation, Koordination und Kooperation betrachtet werden kann. Aufbauend auf dieser Erkenntnis können wir nun auch kooperationsunterstützende Werkzeuge klassifizieren. Je nachdem, auf welcher Ebene ein Werkzeug die Kooperation unterstützt, unterscheidet man Kommunikations-, Koordinations-

und Kooperationswerkzeuge. Im Folgenden sollen deshalb Kommunikations-, Koordinations- und Kooperationsaspekte kooperationsunterstützender Werkzeuge, wie in [Bur94] ausgeführt, betrachtet werden.

9.4.1 Kommunikationsaspekte kooperationsunterstützender Werkzeuge

In diesem Unterabschnitt werden zunächst Kriterien vorgestellt, mit deren Hilfe sich beliebige Kommunikationswerkzeuge klassifizieren lassen. Danach werden drei konkrete Beispiele für Kommunikationswerkzeuge angegeben, auf die dann die Klassifikationskriterien angewandt werden (Abbildung 3).

Die Kriterien, anhand derer sich beliebige Kommunikationswerkzeuge klassifizieren lassen, sind mit ihren Ausprägungen in Abbildung 2 festgehalten. Sie haben folgende Bedeutung:

Zeit: Das Kommunikationswerkzeug kann entweder die gleichzeitige Anwesenheit der Kommunikationsteilnehmer erfordern (synchrone Kommunikation) oder zeitversetzte Kommunikation der Teilnehmer unterstützen (asynchrone Kommunikation)

Weg: Gibt das Verhältnis der Anzahl der sendenden zur Anzahl der empfangenden Kommunikationspartner an.

Medium: Stellt die möglichen Informationstypen dar, die in den Kommunikationsobjekten enthalten sein können.

Erzeugung: kann natürlicher (z.B. Stimme), küstlicher (z.B. Text) oder übersetzter Natur (z.B. Bildbeschreibung) sein.

Quelle: Unterscheidet zwischen Informationen, die zum Zeitpunkt des Sendens entstehen und solchen, die vorab erzeugt werden.

Beziehung: Gibt die Rollenverteilung der Kommunikationspartner an. In einer Master/SlaveBeziehung geht die Initiative von einem Kommunikationspartner aus und alle anderen spielen eine passive Rolle. In Peer to Peer”Beziehungen sind alle Kommunikationspartner gleichberechtigt und können jederzeit selbst Kommunikationsvorgänge auslösen.

Zeit	synchron			asynchron
Weg	1 → 1		1 → n	m → n
Medium	Audio	Video	Text	Grafik
Erzeugung	natürlich		künstlich	übersetzt
Quelle	playback			live
Beziehung	Master / Slave		Peer to Peer	sonstige Rollen

Abbildung 9.2: Morphologischer Kasten: Kommunikation

Beispiele für Kommunikationswerkzeuge sind:

- Ein **email-System**, dient der asynchronen Übermittlung von Nachrichten von einem Sender zu möglicherweise mehreren Empfängern. Sie kann aus unterschiedlichen Medien zusammengesetzt sein, wird jedoch überwiegend zur Übertragung von künstlich erstellten Texten verwendet. Emails werden meist erstellt, um sie sofort zu verschicken; sie können aber auch vorher aufgezeichnete Nachrichten sein.
- **Videokonferenzsysteme** verbinden mehrere gleichberechtigte Teilnehmer miteinander, wobei alle gleichzeitig senden und empfangen können. Es werden sowohl die Bilder der Teilnehmer, als auch deren Sprache live übertragen. Die übertragene Information ist somit natürlich.
- **Virtuelle elektronische Wandtafeln** verbinden mehrere gleichberechtigte Teilnehmer miteinander, wobei alle gleichzeitig über eine virtuelle Zeichenfläche miteinander in synchroner Weise kommunizieren. Die Teilnehmer manipulieren dazu meist Texte oder graphische Objekte, die live während einer Diskussion entstehen.

9.4.2 Koordinationsaspekte kooperationsunterstützender Werkzeuge

Während die Aufgaben von Kommunikationswerkzeugen und die Mechanismen zur Kommunikation bekannt sein sollten und deshalb im vorigen Abschnitt nicht noch einmal ausgeführt wurden, bedarf es hinsichtlich der Aufgaben von Koordinationswerkzeugen und der Mechanismen zur Koordination einer eingehenden Erläuterung. Aus diesem Grunde werden in getrennten Unterabschnitten zunächst die Aufgaben von Kooperationswerkzeugen und einige Mechanismen zur Koordination zusammengefaßt und erklärt. Danach werden auch für die Koordination Kriterien vorgestellt, anhand derer sich beliebige Koordinationswerkzeuge klassifizieren lassen. Diese Kriterien werden dann wieder auf drei konkrete Beispiele für Koordinationswerkzeuge angewandt.

	Zeit	Weg	Medium	Quelle	Erzeugung	Beziehung
E-Mail	asynchron	1→1 1→n	Text Text + ...	künstlich übersetzt	live playback	Peer to Peer
Videokonferenz	synchron	n→m	Video + Audio	natürlich	live	Peer to Peer
elektr. Wandtafel	synchron	n→m	Text + Grafik	künstlich	live playback	Peer to Peer

Abbildung 9.3: Einordnung von Kommunikationswerkzeugen

9.4.2.1 Aufgaben von Koordinationswerkzeugen

Koordinationswerkzeuge dienen der Abstimmung von Teilzielen inklusive der Planung von Aktivitäten, der Rollen- und Ressourcenzuweisung an Subjekte sowie der Überwachung und Zusammenführung der Ergebnisse der Aktivitäten. Diese drei Aufgabenbereiche sollen nun am Beispiel der 'Virtuellen Wochenschau' verdeutlicht werden.

Planung von Aktivitäten

Wie in Abbildung 4 dargestellt, könnte eine Aktivitätenplanung zur Erreichung des Ziels 'Illustrierung eines Textes' folgendermaßen aussehen: Die erste Aktivität des Bildgestalters besteht in der Auswahl von geeigneten Fotos oder Bildern. Als nächstes muß er mit dem Texter des Artikels in Verbindung treten, um die Bildauswahl mit ihm zu besprechen. Ist der Texter mit der Bildauswahl zufrieden, so besteht die nächste Aktivität des Bildgestalters in der Bearbeitung des Bildmaterials (z.B. hinsichtlich Farb- und Größenkorrekturen, Fotomontagen, usw.). Andernfalls muß die Aktivität der Auswahl wiederholt werden. Ist die Bearbeitungsphase abgeschlossen, so ist das Ziel des Bildgestalters erreicht.
Die erste Aktivität, die zur Erreichung des Ziels 'Erstellung des Artikeltextes' nötig ist, könnte gemäß Abbildung 4 im Verfassen des (Roh-)Textes bestehen. Erst wenn dieser entsprechend der Vorgaben der Abteilung Montage formatiert und in einer weiteren Phase Korrektur gelesen wurde, kann das Ziel des Textgestalters als erreicht angesehen werden.
Die Objekte 'Bild' und 'Text' (die ja die erwünschten Endzustände der obigen Teilziele darstellen) werden dann vom Layouter zusammengefügt. An diese Aktivität schließt sich die Phase der Feinabstimmung des Artikels (Layout) an. Erst nachdem diese Aktivität abgeschlossen ist, ist das Gesamtziel der Kooperation (die Erstellung des Artikels) erreicht.
Ein Koordinationswerkzeug muß den oder die Koordinatoren bei der Erstellung solcher Aktivitätspläne unterstützen.

Rollen- und Ressourcenzuweisung

Die Aufgabe der Rollenzuweisung wurde schon im vorigen Kapitel genügend erklärt und motiviert. Es soll hier nur kurz erwähnt werden, wie eine Ressourcenzuweisung für unser Beispiel aussehen könnte. Hier wäre es denkbar, daß innerhalb der Virtuellen Redaktion ein Arbeitsbereich 'Clinton-Artikel' angelegt wird, in dem alle zur Kooperation benötigten Dokumente, Informationen, usw. verwaltet werden. Die Zugriffsrechte auf diese Objekte werden dann durch die Rolle definiert, die jeder Mitarbeiter der

Kooperation innehat.

Überwachung von Aktivitäten und Zusammenführung von Ergebnissen

Um die Kooperation ‘Clinton-Artikel‘ überwachen zu können, muß ein Werkzeug zu jedem Zeitpunkt der Kooperation genau wissen, in welcher Aktivitätenphase des Aktivitätenplans die einzelnen Mitarbeiter der Kooperation sich gerade befinden. Mithilfe dieses Wissens kann das Werkzeug dann vor allem den Zeitplan der Kooperation überwachen und Zeitverzüge an die entsprechenden Mitarbeiter oder Koordinatoren melden.
In unserem Beispiel könnte die Aufgabe der Zusammenführung von Arbeitsergebnissen von einem Werkzeug in folgender Weise erledigt werden: Nachdem auf den Objekten ‘Text‘ und ‘Bild‘ alle nötigen Aktivitäten durchgeführt wurden, könnten diese Objekte vom Werkzeug gesammelt und in einem gesonderten Teils des Arbeitsbereichs ‘Clinton-Artikel‘ innerhalb der Virtuellen Redaktion abgelegt werden. Der Layouter könnte dann zentral auf diese zusammengeführten Arbeitsergebnisse zugreifen.

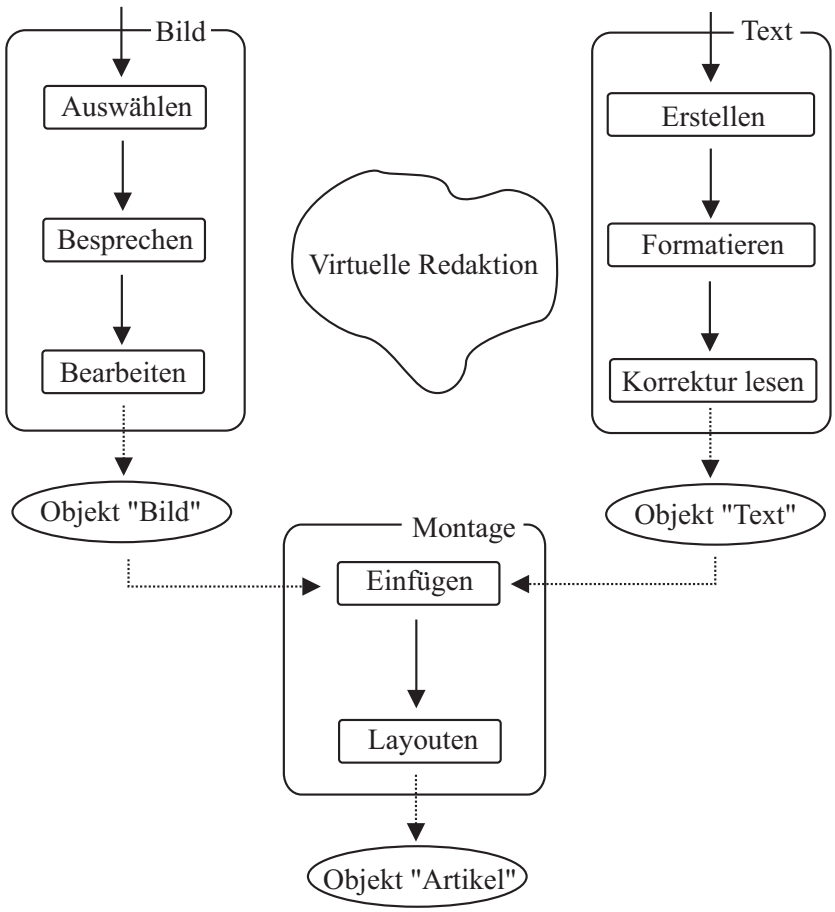


Abbildung 9.4: Beispiel für eine Aktivitätenplanung

9.4.2.2 Mechanismen zur Koordination

In diesem Abschnitt sollen ausgewählte Mechanismen vorgestellt werden, mit deren Hilfe ein Kooperationswerkzeug seine Aufgaben erledigen kann. Die drei hier angesprochenen Problembereiche sind:

- Wie kann man eine Koordination von Aktivitäten, wie im vorigen Abschnitt beschrieben, realsieren? Hierzu werden die Konzepte der Low-Level und High-Level Koordination eingeführt. Dabei wird insbesondere die Protokolltechnik erklärt.
- Wie kann man eine Kooperation bei Bedarf nachvollziehen? Hierzu wird der Mechanismus der Protokollierung vorgestellt.
- Wie können Konflikte, die bei einer Kooperation auftreten, durch ein Kooperationswerkzeug aufgelöst werden? Hierzu werden verschiedene Mechanismen zur Konfliktbehandlung vorgestellt.

Die Mechanismen der Low-Level und High-Level Koordination, der Protokollierung und der Konfliktbehandlung sollen nun der Reihe nach beschrieben werden.

Low-Level und High-Level Koordination

Als Low-Level Koordination wird eine Erweiterung des in Abbildung 1 beschriebenen Kommunikationsmechanismus verstanden. Die Erweiterung besteht darin, daß der Empfänger die aus dem Objektzustand resultierende Information nicht nur wahrnimmt, sondern selbst mit einer eigenen Aktion auf die Wahrnehmung des Objektzustands reagiert.
Ein sehr allgemeines Konzept zur Low-Level Koordination stellen die sogenannten communicative acts dar. Dies sind typisierte Kommunikationsobjekte, die ihren Adressaten zu einer gewissen Aktion veranlassen wollen. Typisiert bedeutet in diesem Zusammenhang, daß die Nachricht des Senders von einem gewissen Nachrichtentyp ist (z.B. Information, Anfrage, Beschwerde). Wird beispielsweise eine Nachricht vom Typ ‘Anfrage‘ verschickt, so erwartet der Sender vom Empfänger der Nachricht, daß dieser ihm umgehend eine Antwort auf die gestellte Frage gibt.
Werden communicative acts im Hinblick auf ein spezielles Ziel zusammengefaßt, so entsteht ein Protokoll. Ein Protokoll ist also eine Folge von genau definierten Kommunikations- und Berechnungsschritten zur Erreichung eines Ziels. Im Zusammenhang mit Protokollen spricht man auch von High-Level Koordination.
Um den Begriff der Protokolle zu veranschaulichen soll wieder unser Beispiel des Aktivitätenplans (Abbildung 4) aufgegriffen werden. Das Werkzeug soll die zur Zielerreichung benötigten Aktivitäten überwachen und Folgeaktivitäten koordinieren. Dazu könnte folgendes Protokoll definiert werden (Auszug):

- Verschicke Nachricht des Typs ‘Information‘ an Bildgestalter, Texter und Layouter: Es kann mit der Kooperation begonnen werden.
- Wenn sich der Bildgestalter oder Texter in der Planphase X befindet und die für diese Aktivität geplante Zeit überschritten ist, dann verschicke Nachricht des Typs ‘Anfrage‘ (oder ‘Beschwerde‘) an die entsprechenden Mitarbeiter.
- Wenn Das Objekt ‘Text‘ vom Texter erstellt und formatiert wurde, dann verschicke Nachricht des Typs ‘Information‘ an Korrekteur: Es kann mit dem Korrekturlesen begonnen werden.
- ...

- Wenn die Teilziele des Bildgestalters und Texters erreicht sind, dann kopiere die entsprechenden Objekte in einen eigenen Bereich und verschicke Nachricht des Typs ‘Information‘ an Layouter: Alle Objekte liegen bereit! Es kann mit der Montage begonnen werden.
- ...

Protokollierung

Ein weiterer Koordinationsmechanismus ist die Protokollierung. Sie wird vorgenommen, um eine Kooperation bei Bedarf nachvollziehen zu können. Dies kann vor allem im Falle von Konflikten (s.u.) oder am Ende einer Kooperation wichtig werden. Ist eine Kooperation beendet, so kann es von Interesse sein, z.B. hinsichtlich der Aktivitätenplanung eine Optimierung vorzunehmen, um wiederkehrende Arbeitsabläufe zukünftig effizienter zu gestalten.
Eine Protokollierung kann hinsichtlich der Größen Objekt, Aktivität und Teilnehmer durchgeführt werden. Dabei sollte in einem Protokolleintrag festgehalten werden, welcher Teilnehmer mit welcher Aktivität eine Zustandänderung eines Objekts herbeigeführt hat. Ein Protokolleintrag sollte zusätzlich mit einem Zeitstempel versehen werden.

Konfliktbehandlung

Wenn Menschen zusammenarbeiten, werden sich Konflikte nie ganz vermeiden lassen. Ein Konflikt tritt dann auf, wenn Teilziele nicht optimal aufeinander abgestimmt sind und so die Erreichung der Teilziele nicht vollständig der Erreichung des übergeordneten Kooperationsziels dienlich sind. Mechanismen zur Konfliktbehandlung sind die Erkennung, die Meldung und Lösung von Konflikten. Während von einem Werkzeug im allgemeinen erwartet wird, daß es Konflikte erkennt und meldet, sollte es hinsichtlich der Auflösung von Konflikten nur eine passive Rolle spielen. In [Bor95] wird darauf hingewiesen, daß es von den am Konflikt beteiligten Personen oft als Ärgernis empfunden wird, wenn das Werkzeug Konflikte selbständig und ohne ihre Beteiligung auflöst. Stattdessen können Möglichkeiten zur Verfügung gestellt werden, die es den Konfliktparteien gestatten, den Konflikt selbst aufzulösen. Zwei solcher Konfliktauflösungsstrategien werden als **bargaining** und **forcing** bezeichnet. Beim bargaining verhalndeln die Gruppen so lange miteinander, bis kein kleinerer gemeinsamer Nenner mehr zu finden ist. Forcing bedeutet, daß ein meist übergeordnetes Subjekt den Konflikt auflöst, indem es den Konfliktparteien vorschreibt, wie fortzufahren ist.
In unserem Beispiel könnte ein Konflikt dadurch entstehen, daß die Textlänge des Clinton-Textes nicht den Vorgaben entspricht. Das Werkzeug könnte diesen Konflikt leicht durch einen Vergleich mit den Vorgaben aufdecken (Erkennung). Es würde diesen Konflikt dann an den Texter und den Layouter weiterleiten (Meldung) und durch das Angebot der Verhandlungsstrategie bargaining bei der Konfliktauflösung behilflich sein.

9.4.2.3 Klassifikationskriterien für Koordinationswerkzeuge

Die in [Bur94] vorgestellten Kriterien, mit deren Hilfe sich beliebige Koordinationswerkzeuge klassifizieren lassen, sind mit ihren möglichen Ausprägungen in Abbildung 5 festgehalten. Sie haben folgende Bedeutung:

Zeit: Ein Koordinationswerkzeug kann die gleichzeitige Anwesenheit der Beteiligten erfordern oder nicht.

Formalisierung: Gibt an, wie formal Benutzer mit dem Werkzeug interagieren. Ein Formalismus schlägt sich auch bei den betrachteten Koordinationsgegenständen und der verwendeten Sprache nieder.

Führung: betrifft die Suche nach Koordinationsentscheidungen. Ein Werkzeug kann dem Benutzer völlig freie Hand lassen, bei der Auswahl und Abfolge von Aktionen, es kann durch ein Hilfesystem die Benutzung des Werkzeugs erleichtern, es kann die Aktionen des Benutzers überwachen und im Fehlerfalle eingreifen, es kann Vorschläge für Benutzeraktionen machen oder es kann den Benutzer hinsichtlich seiner Aktionen stark steuern.

Entscheidungskompetenz: betrifft im Gegensatz zur Führung nicht die Suche nach, sondern das Fällen von Koordinationsentscheidungen.

Koordinationsgegenstand: beschreibt, welche Klassen von Objekten Gegenstand der Koordination sein können.

Anzahl Benutzer: beschreibt, wieviele verschiedene Benutzer direkt mit dem Werkzeug arbeiten können. Die Ausprägung 'n' bedeutet, daß eine festgelegte Anzahl von Personen mit dem Werkzeug arbeiten, während bei einer '¿1' Ausprägung eine beliebige Anzahl von Personen das Werkzeug nutzt.

Rollen: unterscheidet, ob Benutzern mit dem Werkzeug unterschiedliche Rollen zugeordnet werden oder nicht. Die folgenden beiden Kriterien sind nur von Belang, wenn Rollen verwendet werden.

Rollenbezug: Rollen können sowohl objektrelevant, aufgabenrelevant (verschiedene Aufgaben, die einem Subjekt zugewiesen werden) als auch teamrelevant (das Verhältnis der Subjekte untereinander bescheibend) sein. In einem System können mehrere dieser Rollentypen gleichzeitig auftreten. Die Ausprägungen dieses Kriteriums sind also beliebige Kombinationen der Rollentypen.

Rollenzuordnung: Rollen können den Benutzern fest zugeordnet werden oder dynamisch während der Interaktion mit dem Werkzeug vergeben werden.

Es sollen nun drei Beispiele für konkrete Koordinationswerkzeuge betrachtet werden, die anschließend wieder in einer Tabelle (Abbildung 6) anhand der vorgestellten Kriterien klassifiziert werden.

- Ein elektronischer Terminkalender dient der Verwaltung der Termine einzelner Personen und der Abstimmung gemeinsamer Termine in Gruppen. Er kann je nachdem, mit welchen Vollmachten er ausgestattet ist, entweder Vorschläge für die Festlegung gemeinsamer Termine machen, oder für alle Teilnehmer bindende Termine selbstständig festlegen.

- Workflow Management-Systeme strukturieren die Bearbeitung von Vorgängen, indem sie deren Weg durch verschiedene Bearbeiter koordinieren.

- CASE-Tools koordinieren den Prozeß der Softwareerstellung, indem sie, aufbauend auf dem zugrundeliegenden Modell der Softwareentwicklung, die Arbeit der Entwickler in den einzelnen Phasen der Programmerstellung überwachen und unterstützen.

Zeit	synchron					asynchron
Formalisierung	informell	···	strukturiert	···	formal	
Führung	ungeführt	Hilfe	Überwachung	Vorschlag	Steuerung	
Entsch.Kompetenz	keine		Vorschlag		Entscheidung	
Koor.Gegenstand	Koop.Struktur	Ziele	Pläne	Rollen	Ressour.	Ergeb.
Anzahl Benutzer	1	>=1		>1	n	
Rollen	nicht vorhanden				vorhanden	
Rollenbezug	Objektrelevant		Teamrelevant		Aufgabenrelevant	
Rollenzuordnung	statisch				dynamisch	

Abbildung 9.5: Morphologischer Kasten: Koordination

9.4.3 Kooperationsaspekte kooperationsunterstützender Werkzeuge

Es sei noch einmal daran erinnert, daß Kooperation den sich wiederholenden Prozeß der Koordination von Teilzielen und der anschließenden Erreichung dieser Teilziele durch Aktivitäten beschreibt. Darum beinhaltet ein Kooperationswerkzeug auch immer die beschriebene Funktionalität von Koordinations- und Kommunikationswerkzeugen. Darüber hinaus kann ein Kooperationswerkzeug noch übergeordnete Funktionen wahrnehmen, wie etwa die gemeinsame Datenverwaltung oder die Teamverwaltung.
In diesem Abschnitt werden deshalb zunächst Mechanismen zur Daten- und Teamverwaltung besprochen, ehe abschließend der 'Shared Workspace' als Beispiel für ein Kooperationswerkzeug betrachtet wird.

Datenverwaltung

In Kooperationswerkzeugen muß die Datenverwaltung auf kooperativen Zugriff ausgelegt sein. In diesem Zusammenhang kann man folgende Mechanismen zum Zugriff auf gemeinsame Daten anwenden:

- Hinsichtlich des Zugriffs auf Objekte werden in kooperativen Umgebungen **Zugriffskontrollen** zwingend erforderlich. Die Zugriffsrechte kann man dabei wie bereits mehrfach erwähnt durch Rollenzuweisungen an Benutzer regeln. Dabei sollten die Zugriffsrechte in kooperativen Umgebungen aber nicht nur aus generellen Lese- oder Schreibrechten bestehen, sondern auch definieren, welche manipulierenden Operationen ein Benutzer auf dem Objekt ausführen darf.

- Herkömmliche Verfahren zur Synchronisation von Benutzern wie **Transaktionsmechanismen** und Sperrverfahren müssen an kooperative Arbeitsumgebungen angepaßt werden. Diese Anpassung besteht hinsichtlich der Transaktionsmechanismen vor allem in einer Abschwächung der geltenden Konsistenzbedingungen.

	Zeit	Formalisierung	Führung	Entscheidungs-kompetenz	Koordination-gegenstand	Anzahl Benutzer	Rollen	Rollenbezug	Rollen-zuordnung
Termin-kalendar	async.	formal	Steuerung	Vorschlag Entscheidg	Z, P, Ro, Re, E	>=1	ja	Objekt+ Team	dyn.
Workflow Management	async.	formal strukt.	Vorschlag Steuerung	Vorschlag Entscheidg	S, A, P, (Ro), Re, E	>1	ja	Objekt+ Team+Auf-gaben	statisch dyn.
CASE-Tool	async.	formal strukt.	Hilfe Überw.	keine	Z, P, Ro, Re, E	>=1	ja nein	Objekt+ Team+Auf-gaben	dyn.

Abbildung 9.6: Einordnung von Koordinationswerkzeugen

- In kooperativen Umgebungen, in denen mehrere Menschen gleichzeitig ein gemeinsames Objekt bearbeiten, ist genau zu definieren, ob sich die Ausführung einer zurücksetzenden Operation (z.B. der UNDO-Operation) auf die letzte zustandsändernde Aktion des auslösenden Teilnehmers oder auf die letzte Zustandsänderung des Objekts (für die ja auch ein anderer Teilnehmer verantwortlich sein kann) bezieht.

Teamverwaltung

Hinsichtlich der Teamverwaltung ist zu beachten, daß sich in einer kooperativen Umgebung das Team im allgemeinen selbst verwaltet. Neben der Koordination ihrer eigentlichen Aufgaben müssen die Teammitglieder auch noch ihre Interaktionen mit dem Werkzeug koordinieren und abklären, wer welche Koordinationaufgaben wahrnimmt.

Beispiel: 'Shared Workspace'

Als Beispiel für ein Kooperationswerkzeug soll das Konzept des gemeinsamen Arbeitsbereichs (Shared Workspace) angeführt werden. Die Benutzer haben dabei Zugriff auf die Objekte, die sich im 'Shared Workspace' befinden und können diese manipulieren. Dabei wird jede Zustandsänderung, die ein Benutzer an einem Objekt herbeigeführt hat, sofort auch bei allen anderen Teilnehmern sichtbar. Die Aufgabe des Werkzeugs ist es dabei, gleichzeitige Zugriffe der Benutzer zu synchronisieren. Es muß zusätzlich Kooperationsfunktionen hinsichtlich der Teamverwaltung anbieten, um z.B. festzulegen, wer welche Operationen auf Objekten durchführen darf.

9.5 Anforderungen an ein universelles Werkzeug

Im vorigen Kapitel wurden Aspekte von Kommunikations-, Koordinations- und Kooperationswerkzeugen besprochen. Nun soll es darum gehen, aus den besprochenen Konzepten, Anforderungen an ein

universell einsetzbares kooperationsunterstützendes Werkzeug abzuleiten.
Hierzu werden der Reihe nach die Anforderungen an die Kommunikations-, die Koordinations- und die Kooperationsunterstützung eines universellen Werkzeugs vorgestellt. Wiederum sind die Ausführungen aus [Bur94] übernommen.

9.5.1 Anforderungen an die Kommunikationsunterstützung

Die Sichtbarkeit von Kommunikationsobjekten und Kommunikationsvorgängen muß flexibel spezifizierbar sein. Da sich anhand der in Abbildung 2 angegebenen Kriterien und Ausprägungen alle denkbaren Kommunikationswerkzeuge einordnen lassen, sollte ein universelles kommunikationsunterstützendes Werkzeug diese im morphologischen Kasten dargestellten Ausprägungen von Kommunikationsbeziehungen komplett abdecken.

9.5.2 Anforderungen an die Koordinationsunterstützung

Die im vorigen Kapitel beschriebenen Mechanismen zur Koordination sollten so allgemein unterstützt werden, daß beliebige Koordinationswerkzeuge spezifiziert und verwendet werden kön-nen. Insbesondere sollten beliebige Protokolle erstellbar sein und Grundmechanismen zur Konfliktbehandlung angeboten werden.

Die im morphologischen Kasten zur Koordination (Abbildung 5) dargestellten möglichen Ausprägungen von Koordination sollten möglichst komplett abgedeckt werden.

Das Werkzeug sollte die koordinierten Aktivitäten selbständig überwachen können. Hierzu müssen flexible Mechanismen zur Interaktion zwischen Koordination und Aktivitäten bereitgestellt werden.

Es sollte von einem universellen Werkzeug eine Protokollierung durchgeführt werden, um Koordinationsentscheidungen bei Bedarf nachfollziehen zu können. Das Werkzeug sollte ferner das Zurücksetzen von Koordinationsentscheidungen und Aktivitäten unterstützen.

9.5.3 Anforderungen an die Kooperationsunterstützung

Ein universelles Werkzeug sollte die im vorigen Kapitel genannten Konzepte der kooperativen Datenverwaltung berücksichtigen. Bei der Teamverwaltung sollte es insbesondere möglich sein, auch dynamisch veränderliche Teams zu verwalten. Zusätzlich zu diesen Mechanismen sollten Funktionen zur Verwaltung und Überwachung des Gesamtziels (Kooperationsziels) angeboten werden.

9.6 Fazit für die Projektgruppe

Wir haben durch das in Abschnitt 2 vorgestellte universelle Modell rechnergestützter Kooperation eine detaillierte Vorstellung erhalten, was genau unter computergestützter Zusammenarbeit von Menschen zu verstehen ist. Auf dieses Wissen können wir, wie eingangs erwähnt, zurückgreifen, wenn wir kooperative

Aspekte in unser allgemeines Labormodell aufnehmen wollen.
In Abschnitt 4 haben wir Anforderungen an ein universell einsetzbares kooperationsunterstützendes Werkzeug kennengelernt. Falls wir uns dazu entschließen, ein Werkzeug zu entwerfen, das das kooperative Arbeiten von Menschen in virtuellen Laboren unterstützt, dann können uns die Anforderungen an ein universelles Werkzeug wertvolle Anregungen dafür geben, welche Aspekte wir beim Entwurf unseres Werkzeugs berücksichtigen können. Da wir aber nicht jede denkbare rechnergestützte Kooperation unterstützen wollen, sondern nur das gemeinschaftliche Arbeiten von Menschen in virtuellen Laboratorien, müssen wir eine Auswahl aus den möglichen unterstützbaren Aspekten der Kooperation treffen. Dabei wird es unsere Aufgabe sein, zu entscheiden, welche Aspekte im Kontext virtueller Laboratorien wichtig sind und durch das Werkzeug unterstützt werden sollen und welche eher unwichtig sind.

Kapitel 10

Lehr-/Lernsysteme

10.1 Zusammenfassung

Autor: Marco Lindner

In dieser Arbeit wird zunächst definiert, was Lernsoftware ist. Dazu werden die Begriffe Multimedia, Hypertext und Hypermedia erklärt. Es werden Lernkonzepte und Lernformen vorgestellt und diskutiert, inwiefern diese für die virtuellen Labore von belang sind. Darüber hinaus, werden einige wichtige Begriffe aus dem Bereich der Lernsoftware vorgestellt. Im nächsten Abschnitt wird dann die Gestaltung von Lernsoftware hinsichtlich Interaktivität, Abbilder, Hypermedia, kooperativen Lernens und Adaptivität betrachtet. Diese Möglichkeiten bringen Chancen aber auch Probleme mit sich, die im einzelnen besprochen werden.

10.2 Einleitung

Das Ziel der PG ist, die Entwicklung virtueller, naturwissenschaftlicher Labore mit Werkzeugen zu unterstützen. Doch wer soll diese Labore benutzen? Die naturwissenschaftliche Forschung wird auf diese Labore kaum zurückgreifen können, da neue Erkenntnisse selbstverständlich nicht von uns modelliert werden können. Unsere Modelle können sich höchstens auf das beziehen, was heutzutage Stand der Forschung ist. Für Lernende dieser Wissenschaften wären virtuelle Labore interessant. Statt in realen können sie in den virtuellen Laboren Experimente duchführen, und damit den praktischen Teil ihrer Disziplin lernen. Letztendlich wird also mit unserem Werkzeug ein Stück Lernsoftware hergestellt, die bei der Ausbildung von Naturwissenschaftlern benutzt werden kann. Also müßen wir uns mit den Fragen befassen: ,,Was Kennzeichnet Lernsoftware?" und ,,Was muß man beim Entwurf von Lernsoftware beachten?" Die Didaktik ist die Wissenschaft, die erklärt, wie gelernt wird. Deswegen beschäftigt sich diese Arbeit mit einigen Konzepten der Didaktik. Darüber hinaus bietet uns der Computer Möglichkeiten, die man beim konventionellen Lernen nicht hat. Beispielsweise kann man Hypertext, Multimedia und Interaktivität zur Präsentation der Lerninhalte verwenden. Außerdem kann man sich die Vorteile des WWW (wie Unabhängigkeit von Zeit und Ort) zu Nutze machen.

Im folgenden Kapitel wird dargestellt, was Lernsoftware leisten kann und leisten sollte. Es wird zunächst erläutert, was sich hinter den Begriffen Multimedia, Hypertext und Hypermedia versteckt. An-

schließend werden einige Lernkonzepte und Lernformen vorgestellt. Diese sollen Antworten auf die Frage geben, wie man lernt. Darüber hinaus werden verschiedene Begriffe aus dem Bereich der Lernsoftware vorgestellt. Im nächsten Kapitel werden dann Gestaltungshinweise für Lernsoftware hinsichtlich Interaktivität, Hypermedia und Adaptivität, betrachtet. Diese Eigenschaften bieten große Möglichkeiten, allerdings bedingen sie auch einige Probleme, deren Lösungen ebenfalls in diesen Kapiteln skizziert werden sollen.

10.3 Definition von Lernsoftware

10.3.1 Multimedia

Für Multimedia existiert keine allgemein akzeptierte Definition. Schon innerhalb gleicher Fachgebiete, kann es darüber verschiedene Ansichten geben. Im Bereich des Marketings z.B. wird Multimedia wie folgt definiert:
„Multimedia - unter diesem Begriff versteht man die Integration von Text, Graphik, Pixelbildern, Video und Audio” [Old98]
Ralf Steinmetz versteht unter Multimedia:
„Ein Multimedia-System ist durch rechnergesteuerte, integrierte Erzeugung, Manipulation, Darstellung, Speicherung und Kommunikation von unabhängigen Informationen gekennzeichnet, die in mindestens einem kontinuierlichen und einem diskreten Medium kodiert sind.” [RS95] [Old98]
Dabei sind diskrete Medien zeitunabhängige Medien wie Text oder Graphik. Kontinuierliche Medien dagegen sind zeitabhängig wie Audio oder Video.
Es gibt verschiedene Ebenen der Unabhängigkeit der Information. Auf der einen Seite wäre der Videorecorder, mit dem man Bild und Tonfrequenzen abspielen kann. Hier gehören Ton und Bild zusammen, Datenunabhängigkeit ist also nicht gegeben. Auf der anderen Seite genügt die Kombination von DAT-Signalen und computerverfügbarem Text, zum Zweck der Präsentation, der Forderung nach Unabhängigkeit.
Darüber hinaus gibt es die Bedingung der rechnergesteuerten Integration. Ein Textprogramm, mit dem man Tabellenkalkulationen und sogar Videoclips in den Text einbinden kann genügt dieser Forderung nicht, wenn keine Verbindung zwischen den Daten aufgebaut werden können. Ein hohes Maß an Integration liegt vor, wenn eine Änderung im Text eine Veränderung der im Video oder in der Tabellenkalkulation bedingt.
Darüber hinaus sollte auch die Kommunikation zwischen Multimedia-Systemen unterstützt werden. Ausschließlich lokale Multimedia-Systeme sind in Zeiten der globalen Vernetzung ein Schritt zurück.
Die Multimediadefinition von Steinmetz ist sehr strikt. Viele Systeme, die heute als Multimedia-Systeme angesehen werden genügen dieser Definition nicht in allen Punkten.

10.3.2 Hypertext/Hypermedia

Die Konzeption von Hypertext wurde schon 1945 von Bush entworfen [Old98]. Mit Hypertext wird eine Kategorie von (elektronischen) Dokumenten bezeichnen. Diese Dokumente weisen eine nicht-lineare Netzwerkstruktur auf und bestehen aus textuell kodierten Informationseinheiten (den Knoten des Netzes), die durch assoziative Verweisketten (innerhalb und zwischen Dokumenten) verbunden sind. Als Beispiel kann man HTML (Hypertext Markup Language) nennen.

Hypermedia ist eine Erweiterung des Begriffs Hypertext. Während die Knoten der Hypertext-Dokumente ausschließlich aus Text bestehen, können bei Hypermedia auch Multimedia-Objekte integriert werden.

Aufgrund der nicht-linearen Struktur, kann man Hypertext/Hypermedia-Dokumente nicht Seite für Seite betrachten, wie es in linearen Dokumenten (z.B. Bücher) möglich ist. Statt dessen muß man im Dokument navigieren. Dafür gibt es verschiedene Konzepte.
Browsing entspricht dem Stöbern in der Datenbasis. Man kann zwischen ungerichtetem und gerichtetem Browsing unterscheiden. Beim ungerichteten besteht kein konkreter Plan, eine bestimmte Information zu finden. Die Benutzer lassen dich von der Attraktivität des Informationsangebotes leiten. Beim gerichteten Browsing dagegen wird das Dokument zielgerichtet nach einer bestimmten Information durchsucht. Benutzer wählen diejenigen Verknüpfungen, die dieser Information am nächsten scheinen.
Mittels Suchalgorithmen kann eine Information im Dokument gezielt gesucht werden. Voraussetzung ist, daß die Informationsknoten mit Schlagwörtern belegt werden, nach denen gesucht werden kann.
Darüber hinaus besteht die Möglichkeit, Pfaden zu folgen. Durch anklicken einer Weiter-Taste, kann die Datenbasis so Knoten für Knoten durchwandert werden. Trotzdem sollte die Möglichkeit, vom Pfad abzuweichen oder Rücksprünge zu machen erhalten bleiben.

10.3.3 Lernkonzepte

Ein Lernkonzept beschreibt einerseits die Struktur und die Organisationsform in der Lernen stattfindet. Andererseits bietet es eine Orientierung für die Ziel-, Inhalts- und Methodenentscheidung. Sieben Lernkonzepte werden hier vorgestellt. [Old98]

Situated Learning - Situiertes Lernen

Lernen wird als eine Funktion von Aktivität, Kontext und Kultur, in der das Lernen stattfindet definiert. Lernende werden in eine ”community of practice” involviert, die bestimmte Verhaltensweisen und Einstellungen verinnerlicht hat. Lernende bringen sich in die Gemeinschaft ein, und entwickeln sich vom Anfänger zum Experten. Dieses Ansatz kann durch die Meister - Lehrling Situation erweitert werden. Der Lernende beobachtet den Meister in seiner Arbeitsumgebung. Aus diesen Beobachtungen, Ratschlägen des Meisters und selbständig ausgeführten Übungen bildet der Lernende eigene mentale Modelle. Die tutorielle Bedeutung des Meisters wird allmählich ausgeblendet.

Ein virtuelles Labor könnte ermöglichen, daß Lernenden, die nicht weiterkommen, Hinweise gegeben werden, oder daß die von den Lernenden gemachten Fehler korregiert werden. Statt Musterlösungen vorzugeben, könnte das Programm den Kompletten Lösungsweg vorgeben. Damit übernimmt die Software die Rolle des Meisters (Tutors), der Tips gibt, und den man beobachten kann.

Distance Learning - Tele-Lernen

Beim klassischen Fernunterricht erfolgt die Kommunikation zwischen Lerner und Lehrer konventionell, also größtenteils auf brieflichem Weg. Die Wissensverteilung erfolgt primär durch Studienbriefe. Da die Lernenden Ort, Zeitpunkt und Umfang der einzelnen Lernsitzungen selber bestimmen können, spricht man von äußerer Offenheit. Andererseits sind Lernziele, Inhalte, Aufgaben und Lernkontrollen vorgegeben (innere Geschlossenheit). Das erfordert, daß Lernende ihr Lernen selbst organisieren, was nicht trivial ist. Lernende müssen die Fähigkeit zum selbständigen Lernen und Lerntechniken beherrschen.

Darüber hinaus bestehen sehr hohe Anforderungen an die Motivation der Lernenden. Verwendet man moderne Telekommunikationstechniken (E-Mail, Talk, Chat, Videokonferenzen), spricht man vom Tele-Lernen. Durch diese Techniken kann die Kommunikation zwischen Lehrern und Lernenden intensiviert werden. Darüber hinaus besteht die Möglichkeit Kommunikation zwischen den Lernenden herzustellen. Sie stellen fest, daß sie nicht alleine sind mit ihren Problemen, können Erfahrungen austauschen.

Ein Betreuer kann Lernenden Aufgaben zukommen lassen, die sie mit den virtuellen Laboren Lösen. Die Lernenden schicken Protokolle zurück. Es ist sinnvoll, diese Kommunikation über E-Mail abzuwickeln. Darüber hinaus ist es Vorteilhaft, Protokolle einzelner Lernender öffendlich zugänglich zu machen. Diejenigen, die für eine Aufgabe schon Lösungen abgegeben haben, können sich dann mit Lösungsvorschlägen der anderen beschäftigen.

Open Learning - Offenes Lernen

Wie schon beschrieben wird zwischen äußerer Offenheit und innerer Offenheit Unterschieden. Innere Offenheit bedeutet hohe Flexibilität der Lernenden bei der Wahl der Lerninhalte und Ziele. Auch Selbstdiagnose und Leistungsbewertung übernimmt der Lernende. So kann das Lernen an die individuellen Interessen und Fähigkeiten des Lernenden angepaßt werden. Allerdings sprechen Kritiker vom privatierten Lernen, und erworbene Qualifikationen werden im allgemeinen nicht anerkannt.

Gerade für Lernanfänger eignet sich das Konzept der inneren Offenheit aufgrund der oben angesprochenden Probleme überhaupt nicht. Daher sollten die Lernziele der virtuellen Labore vorgegeben sein. Für über die Lernziele hinaus Interessierte sollte es Möglichkeiten geben, weitergehende Lernziele selbst zu definieren. Der gestiegene Informationsbedarf kann beispielsweise durch Verweise auf relevante WWW-Seiten oder Suchmaschinen gesättigt werden.

Collaborative Learning - Kollaboratives Lernen

Eine Gruppe von Lernenden erwirbt gemeinsam und im Wechselseitigen Austausch Kenntnisse und Fertigkeiten. Die Gruppenmitglieder müssen eine Vielzahl gemeinsamer Aufgaben bewältigen. Die Wissenskonstruktion kann durch Diskussionsprozesse unterstützt werden. Lernende erfahren Reaktionen auf eigene Thesen, verteidigen oder verwerfen sie notfalls. Probleme der kollaborativen (kooperativen) Lernens werden im Abschnitt 3.4 behandelt.

Learning By Doing - Handelndes Lernen

Lernenden wird handelnder Umgang mit dem Lerngegenstand ermöglicht. Durch aktive Auseinandersetzung mit dem Lerngegenstand soll die Trennung zwischen Lerngeschehen und Realität ein Stück weit aufgehoben werden. Ein wichtiger Aspekt ist das einsichtige Handeln mit Gegenständen. Bloßen Hantieren (trial & error) ist degegen ineffizient und sollte vermieden werden.

Im virtuellen Labor könnte dem Lernenden ein Arbeitsgerät und einige Aufgaben dazu vorgegeben werden. Der Lernende soll sich jetzt aktiv mit diesem Arbeitsgerät auseinandersetzen, indem er die Aufgaben bearbeitet. Dazu muß er schon vorher wissen, was man mit diesem Gerät machen kann.

Explorative Learning - Entdeckendes Lernen

Der Lernende soll selbst aktiv werden und sich selbständig mit einem Problem auseinandersetzen, Informationen suchen, Annahmen machen und vermeintliche Lösungen evaluieren. Das wird er gemäß seines Vorwissens tun, so daß neue Erkenntnisse in altes Wissen integriert werden können. Entdeckendes Lernen nutzt die leichte Desorientierung bei der Navigation durchs Lehrmaterial. Lernende werden gezwungen, sich aktiv mit der Struktur des Lehrmaterials auseinanderzusetzen.

Das könnte bedeuten, daß Lernende ein Problem vorgegeben bekommen, das sie im virtuellen Labor lösen sollen. Ihre Aufgabe ist es jetzt, sich die entsprechenden Arbeitsgeräte selbständig auszuwählen und mit diesen möglichst das Problem lösen.

Problem-based Learning - Problemorientiertes Lernen

Im Mittelpunkt stehen Problemstellungen, die authentisch sind, allgemeine oder persönliche Brisanz besitzen oder Neugierde und Fragen aufkommen lassen. Dadurch wird das Interesse der Lernenden geweckt und die Erarbeitung von Lösungen motiviert. Es findet eine aktive Auseinandersetzung mit neuen Inhalten statt.

Da solche Probleme in der Regel sehr Komplex sind ist es fast unmöglich, daraus Aufgaben zu formulieren, die im virtuellen Labor tatsächlich bearbeitet werden können.Eine Möglichkeit wäre, das Problem in mehrere Teilaufgaben zu zerlegen, die über einen längeren Zeitraum gestellt werden können.

10.3.4 Lernformen

Eine Lernform ist die pragmatische Betrachtung der Ideen eines oder mehrerer Lernkonzepte. Hier werden drei Lernformen vorgestellt, die bei der geschichtlichen Entwicklung der Lernsoftware Bedeutung hatten. [Old98]

Lernen anhand von Simulationen

Simulationen sind vereinfachte dynamische Modelle von komplexen Prozessen. Eine Simulation besteht zumeist aus einem Modell und einer Präsentation des jeweiligen Modellzustands. Lernende können Parameter konfigurieren oder direkt manipulativ eingreifen und somit spielerisch mit Situationen oder Vorgängen aus der Realwelt umgehen. Allerdings besteht die Gefahr, daß Lernende kritische Konfigurationen nicht identifizieren können und daher gar nicht ausprobieren. Es wäre von Vorteil, Simulationen vorzukonfigurieren, wobei Lernende nicht gezwungen werden dürfen, vorgefertigte Konfigurationen der Reihe nach zu betrachten.

Es erscheint für die virtuellen Labore sinnvoll, vor allem diese Lernform zu verwenden. Experimente verschiedener naturwissenschaftlicher Disziplinen werden simuliert, wobei die Konzepte ”Handelndes Lernen” und ”Entdeckendes Lernen” besondere Beachtung finden werden.

Drill & Practice - Lernen durch Durcharbeiten und Mechanisieren

Lernende werden durch eine Abfolge von Übungen geleitet, um sich Fähigkeiten einzuprägen oder aufzufrischen. Lösungsschemata müssen dafür vorher durchgearbeitet werden oder schon bekannt sein. Meistens wird innerhalb des Schulunterrichts ”Drill & Practice” verwendet, z.B. bei der Nullstellenberechnung quadratischer Gleichungen, die immer nach dem gleichen Schema verläuft.

Im virtuellen Labor könnte ”Drill & Practice” dazu verwendet werden Standartfähigkeiten, die jeder können muß, einzuüben.

Programmierter Unterricht

Der Lerninhalt wird in unabhängige Informationsstücke unterteilt, die dann als Fakten präsentiert werden. Darauf folgt eine Sequenz von Fragen, die überprüfen, ob das Lernziel erreicht wurde. Durch Reaktion auf die Antworten (Strafe, Lob) findet eine Verstärkung des Lernens statt.

Diese Lernform sollte ein virtuelles Labor nicht unterstützen, da reines Faktenlernen kein Ziel des Lernens mir virtuellen Laboren sein sollte.

10.3.5 Begriffsvielfalt der Lernsoftware

An dieser Stelle werden einige Begriffe vorgestellt, die im Bereich der Lernsoftware Verwendung finden. In der Fachliteratur hat sich kein einheitlicher Gebrauch weniger wohldefinierter Begriffe durchgesetzt. Daher wird nicht versucht, Kategorien von Lernsoftware zu bilden, sondern es sollen nur die verwendeten Akronyme vorgestellt und ihre Bedeutung grob eingeordnet werden. [Old98]

Oberbegriffe

Das Akronym CUL steht für computerunterstütztes Lernen bzw. CAL für Computer Aided Learning. Auf der gleichen Ebene ist Computer Aided Instruction (CAI) einzuordnen, welches aber die Betonung auf Schulunterricht legt. Als Industriestandart für alle Aspekte der beruflichen Weiterbildung hat sich dagegen Computer Based Training (CBT) durchgesetzt. Darüber hinaus wird CBT als Synonym für Weiterbildung an Einzelplatzrechnern mit einer sehr rigiden Drill & Practice-Software benutzt.

In Zusammenhang mit dem WWW wird von WBE und WBT (Web Based Education, Training) gesprochen. Zu Anfang standen CML, CMI und CME (Computer Managed Learning, Instruction, Education) für das Konzept der Lehre mit Computern. In neueren Veröffentlichungen tauchen diese aber nicht mehr auf.

Elektronische Skripte/Electronic Textbooks

Im englischen Sprachraum steht Textbook für die vorlesungsbegleitende Literatur, die an deutschen Hochschulen zumeist anhand von Skripten existiert. Zu Anfang des CUL entstanden Electronic Textbooks (ET), die zumeist als simpler ASCII-Text präsentiert wurden. Heutige ET basieren auf Hypertext. Werden diese durch interaktive Elemente ergänzt spricht man von Interactive Textbooks.

Tutorielle Systeme

Einfache Tutorielle Systeme (TS) sind im Prinzip kaum von Interaktive Textbooks zu unterscheiden. Aber sie bilden die Ausgangsbasis für die Domäne der Intelligenten Tutoriellen Systeme (ITS). Diese werden als Expertensysteme für das Lernen entwickelt. Sie bestehen aus einer lernfähigen Wissensbasis mit Fakten und Regelwissen sowie einem lernfähigen Tutor- und Lernermodell. Das Tutormodell entscheidet auf der Grundlage der Eingabe des Lernenden über den Lernweg, die Methode des Lernens und die Art der Präsentation des zu lernenden Stoffes. Das Lernermodell (Abspeicherung der Lernwege und Fehler des Lernenden) erlaubt lernerbezogene Hilfen und Rückmeldungen.

Lernumgebungen

Wenn verschiedene Komponenten von Lernsoftware integriert werden (z.B. ET und ITS) oder Lernsoftware in konventioneller Software integriert ist, spricht man von Lernumgebungen. Abhän-gig davon, welche Art Lernsoftware genutzt wurde spricht man z.B. von Hypermedia Learning Environments (HMLE) oder Web Based Learning Environments (WBLE). Lernsoftware, die Gruppenlernen durch den Einsatz kollaborativer Werkzeuge unterstützt, wird auch zum Rahmen der Lernumgebungen gezählt. Diese wird unter dem Oberbegriff Computer Supported Learning Environments (CSCL) zusammengefaßt. Wenn das Internet als Medium dient, spricht man auch von Internet Based Group Learning (IBGL).

10.4 Gestaltungsmöglichkeiten für Lernsoftware

Im letzten Abschnitt wurde der Begriff Lernsoftware eingeführt, und es wurde gezeigt, was Lernsoftware leisten sollte. Nun wird dargestellt, welche Möglichkeiten es zur Gestaltung von Lernsoftware gibt. Diese Möglichkeiten, die man beim konventionellen Lernen zum Teil nicht hat, bringen große Chancen, aber auch Probleme mit sich. Für einige dieser Möglichkeiten werden im kommenden Abschnitt die Chancen dargestellt. Darüber hinaus werden die Probleme diskutiert und einige Lösungsansätze vorgestellt.

10.4.1 Interaktion und Interaktivität

Der Begriff Interaktion, abgeleitet vom Lateinischen inter = zwischen und agere = handeln, steht für das Miteinander in Verbindung treten zwischen Individuen und sozialen Gebilden. Später wurde dieser Begriff auf den Bereich der Mensch-Computer Interaktion (HCI = Human-computer Interaktion) erweitert. Das bezeichnet sowohl das reale Nutzungsgeschehen zwischen Mensch und Computer, als auch die entsprechende Teildisziplin der Informatik, die sich mit der Beschreibung, Erklärung und Optimierung dieser Vorgänge befaßt. [Haa97]

Wozu soll Interaktivität überhaupt hergestellt werden? Die Interaktivität eines Programms ermöglicht die Auswahl und die Darbietung von Lerninformationen, die den jeweiligen Interessen und Lernbedürftnissen des Lernenden an einer bestimmten Stelle im Lernprozeß entsprechen. Darüber hinaus können Lernende durch interaktive Techniken aktiv in das Lerngeschehen einbezogen werden. [Haa97]

10.4.1.1 Interaktivität für Lernsoftware

Der Begriff Interaktivität beschreibt die Eigenschaften von Software, dem Benutzer eine Reihe von Eingriffs- und Steuermöglichkeiten zu eröffnen. Obwohl es keine von allen Fachdisziplinen und Anwendergruppen anerkannte Klassifikation der Grundformen der Interaktivität gibt, lassen sich für Lernsoftware bestimmte Stufen der Interaktion unterscheiden [Haa97]:

- Zugreifen auf bestimmte Informationen, Auswählen, Umblättern;
- Ja/Nein- und Multiple-Choice-Antwortmöglichkeiten und Verzweigungen auf entsprechende Zusatzinformationen;
- Markieren bestimmter Informationsteile und Aktivierung entsprechender Zusatzinformationen;
- freier Eintrag komplexer Antworten auf komplexe Fragestellungen mit intelligentem tutoriellen Feedback;
- freier ungebundener Dialog mit einem Tutor oder mit Lernpartnern mit Hilfe von Multimedia- und Hypermediasystemen.

10.4.1.2 Interaktions- und Lehr-Lernmetaphern

Es bleibt die Frage, wie Mensch-Computer-Interaktion hergestellt werden soll. Das Ziel ist, daß sich Lernende möglichst intiutiv mit der Software auseinandersetzen können. Daher ist es von Vorteil, wenn sich Operationen am Rechner an realen Handlungen orientieren, die den Berutzern vertraut sind. Dieses kann durch Metaphern erreicht werden.

Anknüpfungspunkt für die Dialog-Metapher ist die natürliche Menschliche Kommunikation zwischen Gesprächs- oder Lehr-Lernpartnern. Ein elektronisches Gegenstück zu einem bekannten physikalischen Sachbereich, wurde in den Interface-Metaphern geschaffen. Anstatt mit abstrakten Objekten auf dem Bildschirm hantieren zu müssen, kann der Benutzter intuitiv mit bildlichen Repräsentanten (Icons) interagieren. Bekanntestes Beispiel ist die Schreibtisch-Metapher, die für viele Oberflächen verwendet wird. Dort kann man mit elektronischen Dokumenten genauso wie mit Papierdokumenten im Büro umgehen. Für den Umgang mit Hypermediadokumenten ist häufig der Rückgriff auf die Buchmetapher festzustellen.

Häufig orientieren sich Interaktionsmetaphern an vertrauten Handlungen und Prozessen im physikalischen Raum, wie z.B. das Durchwandern eines Hauses oder der Einkauf in einem Warenhaus. Kritiker allerdings bezweifeln, daß man die menschliche Raumorientierung auf die Navigation in einem semantischen Hypermedia-Raum übertragen kann. Lernende entwickeln nicht auf jeden Fall globale Pläne für die Interaktion, aufgrund derer sie eine Route im Hypermedia-Raum aufstellen. Viele Aktionen werden erst als Reaktion auf Hinweisreize in einem bestimmten Kontext ausgeführt.

Der Hauptkritikpunkt, daß räumliche Darstellungen den Bildschirm überladen und in ihrer Komplexität den Lerner überfordern, hat zu einer Entwicklung neuer Interaktionsmetaphern geführt, die sich an der zeitlichen Dimension erzählender (narrativer) und dramatischer Ausdrucksformen orientieren. Die klassischen Kunstformen der Erzählung und des Theaters mit ihren gegliederten spannungsreichen Abläufen werden dabei bewußt als Vorbild genommen und metaphorisch auf die Navigation in

Hypermedia-Basen übertragen. Dies geschieht mit dem Anspruch, die Aufmerksamkeit und das Interesse des Lerners zu wecken oder wachzuhalten. Vorgeschlagen wurden sogenannte guides. Dies sind Begleitpersonen, die nach Bedarf Navigationsunterstützung aus verschiedenen Perspektiven geben. Auch dramentheoretische Ansätze können auf die Interaktion mit dem Computer übertragen werde. Kernpunkt ist die These, daß das Engagement des Lernenden durch anregende dramatische Gestaltung im Fluß motivierter Aufmerksamkeit bleiben soll wie bei einem intensiven Theater-Erlebnis.

Zur Konzipierung neuer Lernsysteme wird die Metapher des Geschichtenerzählens vorgeschlagen. Informationen werden anschaulich und adressatenfreundlich narrativ strukturiert, d.h. in Form einer Geschichte wiedergegeben. Dabei kann man sich an den besprochenen Lernkonzepten orientiere. Es wird allerdings davor gewarnt, diese Konzepte unreflektiert zu Übernehmen. Negativbeispiel sind die ermüdenden Seiten-Umblätter-Architekturen, die sich nicht von dem Vorbild des gedruckten Buches lösen konnten.

Künftige Entwicklungen neuer peripherer Ein- und Ausgabewerkzeuge in der Informatik werden völlig neue Lern-Interaktionsmetaphern mit sich bringen. Interaktionstechniken der virtuellen Welt werden es ermöglichen, Lernende unmittelbar in computergenerierte Welten zu integrieren. [Haa97]

10.4.2 Abbilder

Abbilder unterscheiden sich von logischen oder analytischen Bildern (z.B. Diagrammen) darin, daß sie zeigen, wie etwas aussieht. Dieses etwas kann tatsächlich existieren oder nur in der Vorstellung eines Bildautors. Wie bei den Printmedien ist auch bei den Multimedia-Anwendungen ein Trend zu immer mehr Bildern zu beobachten. Das liegt vor allem an der Attraktivität von Bildern für die Nutzer. Mit Abbildern kann man vieles verständlicher ausdrücken als mit reinem Text. Lernende bekommen viel schneller eine Vorstellung von einem Sachverhalt, wenn ein Text durch Bilder unterstützt wird. Bildautoren müssen sich darüber im Klaren sein, was sie mit einem Abbild aussagen wollen, also welche Funktion das Abbild haben soll.

10.4.2.1 Instruktionale Funktionen von Abbildern

Abbilder dienen nicht zur Unterhaltung sondern sie haben in informierenden und instruktionalen Angeboten bestimmte Funktionen. Diese Funktionen sind dadurch bestimmt, was ein Abbild aussagen soll. Drei wichtige Funktionen werden hier vorgestellt [Wei97]:

- Zeigefunktion
- Situierungsfunktion
- Kunstruktionsfunktion

Zur Zeigefunktion

Lernende sollen mit Hilfe von Abbildungen ein deutliches und zutreffendes Bild von einem Gegenstand entwickeln. Mit Gegenstand ist generell das Bildthema gemeint, es kann sich also auch um einen Bewegungsablauf handeln. Lernende sollen anhand dieser Vorstellung folgende Fragen beantworten können:

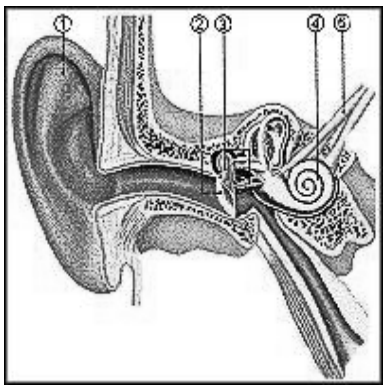


Abbildung 10.1: Zeigefunktion

„Was ist typisch für den Gegenstand?" „Woran erkennt man ihn?" „Was unterscheiden ihn von anderen Gegenständen?" Diese Funktion erfüllen viele Abbilder aus dem Schulunterricht. Die Aufgabe des Zeigens ist anspruchsvoller als es erscheinen mag. Es gilt die Aufmerksamkeit der Lernenden auf die kritischen Merkmale eines Gegenstandes zu richten. Einerseits sollen Lernende eine möglichst vollständige Vorstellung vom Gegenstand entwickeln, andererseits soll Wichtiges von Unwichtigem unterschieden werden. Es hat sich empirisch gezeigt, daß allgemeine Anweisungen an Lernende, einer Abbildung besondere Beachtung zu widmen, wenig fruchten. Statt dessen sind gezielte Hinweise, worauf geachtet werden soll, nötig.

Abbildung 10.2: Situierungsfunktion

Zur Situierungsfunktion

Die Situierungsfunktion erfüllt ein Abbild dann, wenn es dem Betrachter hilft, Detailinformationen in einen Rahmen einzubetten. Häufig werden solche Abbilder im Sprachunterricht eingesetzt. Sie stellen jeweils ein Szenarium bereit, sie aktivieren beim Betrachter Situationsvorstellungen. Situierende Abbildungen aktivieren bei jedem Betrachter eigene Alltagserfahrungen, die reicher als die Bildvorlage sind. Die emotionalen Wirkungen, die Abbildungen zugeschrieben werden, hängen mit dieser Aktivierung von persönlichen Erfahrungen zusammen.

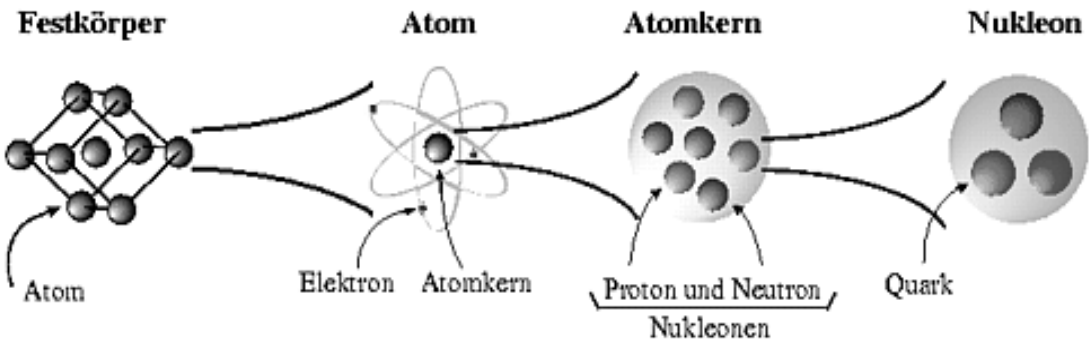


Abbildung 10.3: Konstruktionsfunktion

Zur Konstruktionsfunktion

Komplexere Realitätsausschnitte (z.B. die Funktion eines Motors, der Aufbau der Atome) werden verstanden wenn es der Person gelingt sich ein mentales Modell zu konstruieren. Abbilder können die Lernenden bei dieser Konstruktionsaufgabe unterstützen, indem sie sowohl über Elemente als auch über das Zusammenspiel dieser Elemente visuell informieren. Wegen der verschiedenen Zustandsänderungen lassen sich mentale Modelle am besten durch eine Sequenz von Einzelbildern oder durch Animationen visualisieren.

10.4.2.2 Abbilder als visualisierte Argumente

Absichtsvoll gestaltete Abbilder sind visualisierte Argumente. Sie verlangen vom Betrachter mehr als die naive Blickrichtung Fenster zur Welt. Aufgrund der üblichen Sehgewohnheiten wird der Informationsgehalt von Abbildern immer wieder unterschätzt. Es gilt, die Abbilder so zu gestalten, daß das visualisierte Argument optimale Chancen hat, vom Betrachter extrahiert zu werden. [Wei97]

Gestaltung von Abbildern mit Zeigefunktion

Für die Zeigefunktion gilt es, per Darstellungscodes (Schattierung, Perspektive, Signalfarbe etc.) und Steuerungscodes (Pfeile, Größenverzerrungen, Umrandungen etc.) -eventuell auch mit zusätzlicher Textunterstützung- die Wahrnehmung der als relevant definierten Gegenstandsmerkmale zu sichern. Bei der Darstellung eines Ohres, wird die unwichtige Ohrmuschel klein und kaum schattiert dargestellt. Der Innenohrbereich, der gezeigt werden soll, wird schattierter und detaillierter dargestellt. Darüber hinaus kann durch Beschriftung die Aufmerksamkeit auf die Gehörknöchelchen gelenkt werden. So wird gerährleistet, daß Betrachter das wesentliche am Bild wahrnehmen. Beim vorliegenden Beispiel ist das nicht optimal gelungen. Die Ohrmuschel ist noch zu auffällig. Trotzdem wird die Darstellung des Innenohrs vom Betrachter wahrgenommen. Für die Erfüllung der Zeigefunktion ist das realistische Abbild meist weniger geeignet als eine Abbildung, die das Wesentliche heraushebt. Beispiele für solche Abbildungen findet man häufig in Anatomie- oder Pflanzenbüchern.

Gestaltung von Abbildern mit Situierungsfunktion

Bei der Situierungsfunktion stellt sich vor allem die Frage nach dem Grad des Realismus. Realistische und detaillierte Abbilder situieren zwar am wirkungsvollsten, laufen aber Gefahr, daß Details nicht mit den Erfahrungen des Betrachters übereinstimmen. Außerdem sind detaillierte Situationsbilder kulturgebunden und veralten schnell. Daher werden situierende Abbilder häufig bewußt reduziert Dargestellt.

Gestaltung von Abbildern mit Konstruktionsfunktion

Bei solchen Bildern ist besonders das Komplexitätsproblem zu lösen. Es gilt, das Bildmaterial zur Konstruktion eines mentalen Modells so zu proportionieren, daß die Betrachter es optimal nutzen können. Dabei muß allerdings darauf geachtet werden, daß die Makrostruktur nicht verloren geht. Abbilder, die zur Konstruktion eines mentalen Modells dienen sollen, bedürfen im besonderen Maße der Sprachlichen Umsetzung. Der Grund liegt darin, daß sich per Sprache Beziehungen zwischen Elementen eines Modells präziser und differenzierter ausdrücken lassen als mit piktoralen Mitteln. Dies ist besonders bei Analogiebildern nötig. Wenn Abbilder per Analogie verwendet werden (z.B. die Abbildung des Planetensystems zur Veranschaulichung des Atomaufbaus) gibt es immer nur einige strukturelle oder Funktionale Korrespondenzen, die zutreffen. In anderer Hinsicht hinkt der Vergleich.

10.4.2.3 Abbilder und Multimedia

Gegenüber Printmedien erweitert Multimedia die Gestaltungsmöglichkeiten für Abbilder ganz erheblich. Die Einbeziehung des auditiven Sinneskanals, von Bewegtbildern und von Interaktivität wird in diesem Abschnitt deshalb noch kurz vorgestellt. [Wei97]

Audio: Bildlegenden und Bilderläuterungen per Audio zu präsentieren, hat mehrere Vorzüge. Zum einen wird der Bildschirm vom Text entlastet, zum anderen muß der Betrachter nicht Blicksprünge vom Text zum Bild und zurück ausführen. Ein weiterer Vorzug von Audio ist die Ergänzung situierender Abbilder durch Orginalsprache oder Orginalgeräusche. Schließlich kann Audio für bestimmte Signaltöne genutzt werden, die die Interaktivität eines Anwenders mit dem Lernangebot begleiten und steuern.

Bewegtbilder: Da vieles mit Standbildern nur unzulänglich dargestellt werden kann, kann man in Multimedia-Anwendungen auf Bewegtbilder zurückgreifen. Bewegtbilder bringen das Risiko des Overloads mit sich. Durch Verlangsamung, oder mentale Vorbereitung, kann dieses Risiko gesenkt werden.

Interaktivität: Es ist möglich zu Abbildern oder Teilen von Abbildern Beschriftungen oder zusätzliche Informationen abzurufen. Auch akustische Kommentare oder ein höherer Detaillierungsgrad sind vorstellbar.

10.4.3 Hypertext und Hypermedia

Die Begriffe Hypertext und Hypermedia sind im Abschnitt 2.2 bereits vorgestellt worden. An dieser Stelle sollen sowohl die Vorteile, als auch die Probleme, die diese Konzeptionen mit sich bringen, erläutert werden. Einige Ansätze zur Lösung dieser Probleme werden darüber hinaus besprochen.

10.4.3.1 Vorteile und Chancen hypermedialer Lernsoftware

Die Vorteile von Hypermedia, kann man von zwei Standpunkten aus betrachten. Vom technischen Standpunkt aus kann man die Vorteile aus Sicht der Entwicklung und Wartung anführen, vom didaktischen Standpunkt aus die Vorteile aus Sicht der Lernenden. [Old98]

Vorteile aus Sicht der Entwicklung und Wartung

- modulare Erweiterbarkeit, aufgrund der nicht-linearen Netzwerkstruktur
- der kurze Produktionszyklus vom Design neuer Module bis zur Veröffentlichung
- die einfache Integration von Simulationen und Animationen
- die Struktur eines komplexen Wissensgebietes kann (sofern didaktisch aufbereitet) direkt übernommen werden und muß nicht in ein lineares Konzept gebracht werden
- die weitgehende Eliminierung von Redundanz und damit eine gesteigerte Konsistenz innerhalb des Systems, da von jeder Stelle auf benötigte Informationen Zugegriffen werden kann und sie nicht mehrfach vorgehalten werden müssen
- die Möglichkeit, daß Lernmaterialien auf individuelle Lernsituationen verschieden reagieren können.

Vorteile aus Sicht der Lernenden

- transparent, semantische Verknüpfungen der Informationsknoten durch den nicht-linearen Charakter von Hypermedia
- das große Potential der möglichen Interaktivität läßt Hypermedia besonders interessant für Lernzwecke erscheinen
- ein hohes Maß an Selbststeuerung
- Plattformunabhängegkeit für Lernende und damit die Möglichkeit, persönlich beforzugte Konfigurationen zu nutzen
- die Vermutung, daß Lernerfolge durch die nicht-lineare Netzwerkeigenschaften von Hypermedia begünstigt werden können, sofern gewisses Grundwissen vorhanden und die Motivation der Lernenden sehr hoch ist

10.4.3.2 Probleme bei der Verwendung hypermedialer Lernsoftware

Auch die Probleme hypermedialer Lernsoftware kann man von diesen beiden Standpunkten aus sehen. Allerdings werden die Probleme der Lernenden häufig unterschätzt. Einige Experten zweifeln diese Probleme sogar an. [Old98]

Probleme bei der Entwicklung und Wartung

- Werkzeuge für das Angebot und die Entwicklung von Hypermedialer Lernsoftware sind noch nicht ausgereift
- es werden zum Teil hohe Anforderungen an das Wissen über den Umgang mit den technischen Aspekten der benutzten Software gestellt
- die nicht-lineare Netzwerkstruktur erschwert es, die Korrektheit aller Verknüpfungen zu gewährleisten

Probleme der Lernenden

Zwei Grundtypen von Lernproblemen treten beim hypermedialen Lernen auf. Desorientierung und kognitive Überlast. [Ter97]

Das Problem der Desorientierung wird häufig mit ”Lost in Hyperspace” bezeichnet. Es bezieht sich auf die Navigation in Hypermedia-Systemen. Probleme bei der Navigation in Hypertext-Basen betreffen den fehlenden Überblick der Nutzer über den aktuellen Standort der eigenen Bearbeitung eines Hyperdokuments im Hinblick auf das Gefüge der in der Hypertext-Basis verknüpften Information. Sie betreffen ferner die Unkenntnis darüber, auf welchem Wege und mit welchen Mitteln der Zugriff auf eine bestimmte Information erfolgen kann, von der man weiß, daß sie in der Datenbasis enthalten ist. Das Auftreten von Navigationsproblemen ist darauf zurückzuführen, daß es Benutzern ohne Hilfen nur schwer gelingt, sich eine Vorstellung von der Organisationsstruktur der Datenbasis zu machen. Mangelnde Kenntnis über vorhandene Navigationsmöglichkeiten und deren adäquate Anwendung können weitere Ursachen für Navigationsprobleme darstellen. Navigationsprobleme nehmen in der Regel dann zu, wenn Komplexität und Unstrukturiertheit der Datenbasis steigen. Leichte Desorientierung im Hyperdokument ist teilweise aber auch von den Autoren der Lernsoftware gewollt. Lernende sind gezwungen sich genauer mit dem Dokument auseinanderzusetzen. Versuchspersonen in Lern-Experimenten geben oft zu Protokoll, daß sie glauben, bestimmte Informationen nicht aufgenommen zu haben und daß sie im Netzwerk die Orientierung verloren heben. In Wissenstests haben diese Personen allerdings gut abgeschnitten.

Das Problem der kognitiven Überlast kann wie folgt beschrieben werden. Um mit Hypermedia-Systemen effektiv zu lernen, ist es erforderlich, daß man im Gedächtnis behält, welche Informationsknoten schon aufgesucht wurden, auf welchem weg man zu ihnen gelangt ist, welchen Inhalt diese hatten, welche Informationen noch aufgesucht werden sollten, welche Möglichkeiten der Navigation zur Verfügung stehen, welche Funktion die einzelnen Navigationsmittel erfüllen etc. Dies erfordert zusätzliche Gedächtniskapazität, Aufmerksamkeit und Fähigkeiten zur metakognitiven Kontrolle, die häufig nicht zur Verfügung stehen. Lernende Können durch eine solche kognitive Überlast von einer tieferen Informationsverarbeitung abgehalten werden.

10.4.3.3 Navigations- und Orientierungshilfen

Um den Navigationsproblemen entgegen zu treten ist eine Reihe von Strategien entwickelt worden. Einige werden hier kurz beschrieben. [Haa97]

- Graphische Browser geben in der Form von Netz- oder Baumstrukturen einen globalen Überblick oder einen Überblick über einen Ausschnitt einer Hypermedia-Basis

- Fischaugensichten geben analog zu einer Linse mit großem Winkel eine detaillierte Sicht der nahen Umgebung und eine weniger detaillierte Sicht der weiteren Umgebung
- Leseprotokolle enthalten eine Aufzeichnung der bisher angesteuerten oder bearbeiteten Knoten. Sie ermöglichen einen direkten Zugriff auf diese Knoten oder schrittweise Zurückverfolgen des eigenen Lernpfades
- Breadcrumps ermöglichen die automatische Kennzeichnung bereits bearbeiteter Teile einer Hypertext-Basis, um unfreiwilliges neues Bearbeiten zu vermeiden
- Lesezeichen (Bookmarks) kennzeichnen subjektiv wichtige Bereiche des Lernenden und ordnen eine umfangreiche Hypertext-Basis nach persönlichen Lernbedürftnissen
- Thump taps können von Lernsystem-Autoren eingefügt werden, um relevante Lernbereiche besonders hervorzuheben
- Pfade können das völlig freie Navigieren einschränken

Folgende fünf Fragen sollten Lernende in einem guten Hyperdokument jederzeit beantworten können [Old98]:

- Wo bin ich?
- Wie bin ich hier hingekommen?
- Was kann ich hier tun?
- Wo kann ich hingehen?
- Wie komm ich da hin?

10.4.4 Kooperatives Lernen

Da die virtuellen Labore ans Netz angebunden sind, wird es möglich sein, daß sich Lernende, die über die ganze Welt verteilt sind, zu einer Arbeitsgruppe zusammenschließen. Dadurch treten einige Probleme auf. [FWH97]

10.4.4.1 Problembereiche computerunterstützten kooperativen Lernens

Mangel an sozialer Präsenz

Soziale Präsenz der Kommunikationsteilnehmer ist die differentielle Eigenschaft von Kommunikationsmedien. Diese können danach unterschieden werden, wieviel Kommunikationskanäle sie zur Verfügung stellen und wie gut sie somit Informationen über verschiedene soziale und nonverbale Hinweisreize übermitteln (z.B. Gesichtsausdruck, Blickrichtung, Kleidung). Die computergestützte Kommunikation stellt im Vergleich zur Face-to-Face-Situation sehr viel weniger Kommunikationskanäle zur Verfügung und bietet somit ein geringeres Ausmaß an sozialer Präsenz, so daß das Gefühl, in einer Kommunikationssituation involviert zu sein, bei den Kommunikationspartnern gering sein kann. Dadurch wird die

Kommunikation meist unpersönlicher und aufgabenorientierter. Da auch Hinweise auf den Status der Akteure fehlen, fallen damit zusammenhängende Interaktionsmuster weg. Statusgeringere Teilnehmer haben größere Kommunikationsanteile, wodurch es zum egalitären Austausch kommt. Andererseits fehlt das soziale Feedback, wodurch soziale Normen weniger bindend erscheinen, ungehemmtes Verhalten und stärkere Selbstbezogenheit auftreten.

Fehlende Gruppenkoordination

Gruppenkoordination bezieht sich darauf, wie die Aktivitäten einzelner Gruppenmitglieder oder Subgruppen integriert und harmonisiert werden können, um ein gemeinsames Produkt zu erzielen. Koordinationsmaßnahmen bestehen zum einen darin, die Gruppenarbeit in spezifische Beiträge von Gruppenmitgliedern bzw. Subgruppen aufzugliedern, die wiederum in den gesamten Arbeitszusammenhang eingepaßt werden müssen. Zum anderen müssen wegen der unterschiedlichen Erstellungszeiten der Arbeitsbeiträge und eventueller Verzögerungszeiten aufgrund asynchroner Kommunikation zeitliche Abstimmungen erfolgen. Bei kooperativer Arbeit sollten Maßnahmen zur Gruppenkoordination explizit ergriffen werden. Gruppenstrukturen, die Koordinationsmaßnahmen unterstützen könnten, insbesondere Rollenverteilungen und entsprechende Interaktionsmuster, stellen sich wegen der mangelnden sozialen Präsenz nicht so leicht ein. Grundsätzlich sind zwei formen von Interaktionrregeln zu unterscheiden: technologisch-basierte und sozial-basierte Regeln. Technologische Vorgaben strukturieren den Kommunikationsprozeß und unterstützen weniger erfahrene Nutzer. Sozial basierte Interaktionsregeln werden von der Gruppe selbst entwickelt. Sie sind weniger restriktiv was dem Arbeitsstil erfahrener Gruppen zugute kommt.

Fehlende Abstimmung über gemeinsamen Wissenshintergrund

Damit die Kommunikation in kooperativen Lern- und Arbeitsgruppen gelingt, ist ein bestimmter Anteil gemeinsam geteilten Wissens grundlegend: Jedes Gruppenmitglied hat - damit es eine Nachricht verständlich formulieren kann - eine Vorstellung von dem Wissen des Empfängers und davon, daß dies dem Empfänger bewußt ist. Dieser gemeinsame Wissenshintergrund geht auf Quellen zurück, die in computerunterstützten kooperativen Arbeitsgruppen nur bedingt verfügbar sind.

Überangebot an Information

Informationsüberlastung bei computergestützter Gruppenarbeit erwächst aus den technologischen Möglichkeiten moderner elektronischer Medien, große Nachrichtenmengen zu erzeugen und darzubieten. Die Gruppenmitglieder können Nachrichten in beliebiger Anzahl und Länge generieren. Ihnen stehen darüber hinaus eine Vielzahl weiterer Informationsquellen zur Verfügung: Informationen zur Grupensituation und zur Lern- und Arbeitsgeschichte der Gruppe, z.B. Aufzeichnungen von Interaktionsprotokollen, Informationen über Konferenzteilnehmer etc.. Außerdem ist häufig der Zugriff auf externes Informationsmaterial möglich. Auf diesem Hintergrund besteht beständig die Gefahr, daß durch die auftretenden Informationsmengen die Verarbeitungskapazität eines jeden Mitglieds überlastet und die Gruppenarbeit beeinträchtigt wird. Solche Informationsbelastungen, können zur ungenauen Beantwortung von Nachrichten oder sogar zur Nichtbeachtung führen. Erfahrene Nutzer selegieren jedoch Nachrichten gezielt und schätzen die Nachrichtenmenge sogar positiv ein.

Fehlende Nachrichtenverbundenheit

Mangelnder Bezug von Nachrichten, die die Gruppenmitglieder untereinander versenden ist ein auffälliges Problem in computerunterstützten kooperativen Arbeitsgruppen. Es ergeben sich sowohl zeitliche Verzögerungen, als auch ungenügende inhaltliche Bezüge von Nachrichten. Dar Problem wird deutlich, wenn man sich das Kummunikationsgeschehen als eine Abfolge ineinandergreifender Kommunikationszyklen vorstellt. Jeder Zyklus umfaßt dabei eine Reihe von Elementen: Erstellen und editieren einer Nachricht, übersenden, empfangen, Empfang bestätigen und beantworten einer Nachricht. Solche Kommunikationszyklen glatt und ineinandergreifend zu realisieren, ist bei computergestützter Kommunikation häufig schwierig. Dies gilt vor allem für asynchrone Kommunikation, bei der die einzelnen Kommunikationselemente und -zyklen zeitlich sehr stark verzögert sein können: Zum Beispiel kann das zeitliche Ausbleiben einer Empfangsbestätigung für eine Nachricht beim Sender Unsicherheit erzeigen und zur Übersendung weiterer Nachrichten führen, ohne daß der erste Nachrichtenzyklus abgearbeitet wurde.

10.4.4.2 Lösungsansätze

Die Probleme computerunterstützter Kollaboration können durch bestimmte Funktionen von Groupware-Systemen und mögliche Maßnahmen biem Einsatz dieser systeme gemindert werden. Entsprechende Lösungsansätze werden im folgenden unter zwei Aspekten vorgestellt. [FWH97]

Lösungsansätze bezüglich der sozialen Dimension

Die sozialen Dimension bezieht sich auf die Wahrnehmung der Teilnehmer untereinander und ihr gegenseitiges Verständnis. Diese Dimension umfaßt Probleme der sozialen Präsenz, des gemeinsamen Wissens und der Gruppenkoordination.

Jeder Teilnehmer könnte seine personal view entwerfen. Dabei wählt er ein Portraitfoto für sich aus und gestaltet ein eigenes Monogramm, mit dem er künftig seine Nachrichten signiert. Ferner stellt er eine Liste mit den Aufgaben zusammen, die er in der Gruppe übernimmt und macht Angeben zu seinem Aubildungs- und Wissenshintergrund.

Um soziale Beziehungen zu unterstützen, kann ein Konferenzmodul eingerichtet werden, das die Möglichkeit für informelle, persönliche Mitteilungen bietet (z. B. Cafeteria). Der dadurch angeregte nicht aufgabenbezogene Austausch hat gleichzeitig den Vorteil, die themengebundene Gruppenarbeit von zusätzlichen persönlichen Beiträgen zu entlasten.

Um fehlende nonverbale Kommunikationsmöglichkeiten zumindest Ansatzweise ausgleichen zu können, können Emoticons, wie z. B. Icons mit unterschiedlichen Gesichtsausdrücken, die der Teilnehmer anklicken kann, um seine jeweilige Stimmung zu signalisieren, verwendet werden.

Zur Förderung des Wissenshintergrunds kann eine message history zu jeder verschickten Nachricht dienen. Jeder Teilnehmer kann sich darüber informieren, von wem die Nachricht stammt, und wer sie schon gelesen hat. Konferenzarchive bieten jedem Teilnehmer die Möglichkeit, sich zu vergegenwärtigen, was bereits diskutiert und verarbeitet wurde. Entsprechende Informationen sollten regelmäßig auf den neusten Stand gebracht werden.

Die Methode des strict WYSIWIS (What You See Is What I See) stellt sicher, daß alle Teilnehmer den gleichen Bildschirm vor sich haben. Nachrichtensender können sich darauf verlassen, daß der

Empfänger dieselben Informationen vor sich hat wie man selbst. Der Nachteil des strikt WYSIWIS ist, daß Änderungen die ein Teilnehmer vornimmt, für alle sichtbar werden, was zu einiger Verwirrung führen kann.

Unter Umständen Kann es hilfreich sein, den Teilnehmern unterschiedliche Rechte zu vergeben. Zum Beispiel bekommen die Teilnehmer für die Arbeiten anderer Mitglieder nur Leserechte. Auch vorstellbar ist, daß ein Moderator Zusatzrechte bekommt. Er kann Beiträge anderer löschen, oder neue Mitglieder aufnehmen.

Die Durchführung einer Abstimmungsrunde kann dabei helfen, sich relativ schnell über das weitere Vorgehen zu einigen. Dazu sollte das System Abstimmungsmechanismen (Voting Tools) bereitstellen.

Und der Zugang zu Dokumenten sollte geregelt werden. Wenn zwei Mitglieder das gleiche Stück bearbeiten, kann es zu Inkonsistenzen kommen.

Lösungsansätze bezüglich des Informations- und Nachrichtenaustausches

Die stärker technologische Dimension des Informations- und Nachrichtenaustausches bezieht sich auf die potentielle Informationsüberlastung und mangelnde Nachrichtenverbundenheit.

Um die Teilnehmer nicht mit Informationen zu überschwemmen, bietet sich an zwischen öffentlichen und privaten Mitteilungen zu trennen.

Die Methode des relaxed WYSIYIS ermöglicht es jedem Teilnehmer, nur die Fenster zu öffnen, die für ihn im Moment von Belang sind. Ferner kann er auf private Daten zugreifen, ohne daß diese für die anderen sichtbar werden.

Die Software sollte Informationsfilter bereitstellen, um eine gezielte Nachrichtenauswahl über Schlüsselwörter zu ermöglichen.

Um Nachrichtenverbundenheit zu gewährleisten, sollte jede Konferenz in verschiedene Themen (threads) unterteilt werden. So ist ein klarer Bezug der Beiträge untereinander zu gewährleisten.

10.4.5 Adaptivität und Adaptierbarkeit

Benutzer multimedialer Lehrnsoftware unterscheiden sich hinsichtlich des Ausmaßes an Unterstützung, die sie benötigen, um beabsichtigte Lernziele erfolgreich und mit vertretbarem Aufwand erreichen zu können. Dabei ist es möglich, daß der Unterstützungsbedarf mit der Zeit abnimmt. Durch angemessene Systemanpassungen (Adaptionen) kann die Lernerfreundlichkeit der Lernsoftware erhöht werden. [Leu97]

10.4.5.1 Adaptives Lernen

Die Kunst der Lehrens besteht darin, für eine optimale Passung zwischen dem gegebenen Unterstützungsbedarf der lernender Person und dem in der Lehr-Lernsituation zur Verfügung gestellten Unterstützungsangebot zu sorgen: Selbständige Lernende könnten durch zu viel Unterstützung eingeengt werden (when teaching kills learning). Im Extremfall könnte es genügen, den Lernenden mit einer neuen Komplexen Situation zu konfrontieren, die es langfristig und überdauernd zu meistern gilt. Alles andere wird vom

Lernenden selbst geleistet. Im anderen Extremfall von höchsten Ausmaß unselbständigen Lernens wäre es erforderlich ein großes Unterstützungsangebot bereitzustellen. Zwischen diesen beiden Extremen muß ein geeigneter mittlerer Weg gefunden werden.

Selbstverständlich ist es auch für Lernsoftware sinnvoll, adaptives Lernen zu unterstützen. Schließlich haben auch die Benutzer dieser Systeme unterschiedliche Eigenschaften, möglicherweise ändern sich diese Eigenschaften auch mit der Zeit. Nachfolgend werden zwei Konzepte, adaptives Lernen am Rechner zu verwirklichen, diskutiert:

- Adaptierbarkeit
- Adaptivität

10.4.5.2 Adaptierbarkeit

Ein System ist dann adaptierbar, wenn es durch externe Eingriffe an veränderte Bedingungen angepaßt werden kann. Ein alltägliches Beispiel wäre der verstellbare Autositz. In der Instruktionspsychologischen Literatur wird diese Art der Adaption Makro-Adaption genannt. Die Anpassung des Lernsystems wird in größeren zeitlichen Abständen vorgenommen, mindestens zum Beginn der Lehreinheit. Daher können solche Lernsysteme auch nur an die Eigenschaften der Lernenden adaptiert werden, die wenig veränderlich sind, wie Charakter, Intelligenz, Interessen, Lernstilen etc.

Man kann beispielsweise zwischen Visualisierern und Verbalisierern unterscheiden. Die einen knnen ausgiebig Illustrierte Texte besser verarbeiten und verstehen, die anderen konzentrieren sich mehr auf den Text, wenn sie sich mit einem Dokument auseinandersetzen. Multimediasysteme, die Graphik und Text unterstützen, knnen dann an die jeweiligen Vorlieben der Betrachter adaptiert werden.

10.4.5.3 Adaptivität

Ein System ist dann adaptiv, wenn es sich an veränderte Situationen anpassen kann. Die elektronische Motorsteurung moderner PKW ist ein alltägliches Beispiel für ein adaptives System. Die zu Beginn vorgenommene Anpassung wird andauernd ünbeprüft und aktualisiert. Es werden die Eigenschaften der lernenden zur Adaption herangezogen, die veränderlich sind.

Einige typische Beispiele sollen hier vorgestellt werden

Adaption des Instruktionsunmfangs und der Lernzeit: Im Sinne zielgerichteten Lernens sollte so lange unterrichtet werden, bis Lernende das Ziel erreicht haben. Zu diesem Zweck muß der aktuelle Wissensstand in kurzen Zeitabständer wiederholt diagnistiziert werden. Anhand dessen wird entschieden, ob noch weitergelernt werden soll oder ob das nächste Ziel angestrebt werden kann.

Adaption der Instruktionssequenz: In Abhängigkeit davon, welche Fehler Lernende bei der Bearbeitung einer Aufgabe machen, werden sie zu einer ganz bestimmten Folgeaufgabe verzweigt.

Adaption der Aufgaben-Präsentationszeit und adaptive Antwortbegrenzung: Ziel ist es, die Lernenden zu Beginn des Übens möglichst wenig Fehler machen zu lassen. Dies läßt sich dadurch erreichen, daß man, wenn die Bearbeitung einer Aufgabe fehlerhaft war, die für die nächste Aufgabe zur Verfügung stehende Zeit reduziert und erst nach richtigen Aufgabenlösungen wieder verlängert.

Adaption der Aufgabenschwierigkeit: Lernende gelangen durch richrige Antworten unmittelbar zur nächst höheren Schwierigkeitsstufe. Eine falsche Antwort führt dazu, daß man zurückgestuft wird.

Adaptive Hilfen beim entdeckenden Lernen: In einer Reihe von Experimenten konnte nachgewiesen werden, daß beim entdeckenden Lernen mit computersimuliersen Planspielen vor allem dann spielübergreifendes Wissen gelernt wird, wenn die zur Bewältigung der jeweiligen Problemsituation erforderliche Hintergrundinformation den Lernenden explizit verfügbar ist, und wenn auf genau diese Informationsquelle adaptiv aufmerksam gemacht wird. Der Hinweis sollte dann erfolgen, wenn erkenntlich ist, daß die Information in der Situation nützlich ist und von den Lernender bisher noch nicht zur Kenntnis genommen wurde.

Adaptive Definition neu zu lernender Begriffe: Es bietet sich an, neue Wissensinhalte adaptiv in genau den Begriffen zu erklären, die kurz zuvor erworben wurden. Die Anwendung der etwas älteren Wissensbestände wird damit andauernd trainiert, und neue Inhalte werden end mit dan alten verbunden.

Adaptiver Informationszugriff in Hypertextsystemen: Hier geht es um die Frage des online-dynamischen Verbindens der Informationseinheiten eines umfangreichen Hypertextes (Dynamik linking) Unter Einbeziehung von online-erfaßbaren Eigenschaften des Informations-Suchverhalten eines Benutzers. Die grundlegende Überlegung ist die, daß dem Benutzer automatisch diejenigen Informationseinheiten zur Einsicht angeboten werden sollten, die möglichst viel mit dem gemeinsam haben, für das er sich Augenblicklich zu interessieren scheint.

10.5 Abschließende Anmerkungen

Zunächst wurde gezeigt, was Lernsoftware ausmacht. Multimedia, Hypertext und Hypermedia sind wichtige Begriffe, die erläutert werden mußten. Es wurden Lernkonzepte und -formen vorgestellt und deren Bedeutung für die virtuellen Labore diskutiert. Vor allem sollten die Konzepte handelndes Lernen und entdeckendes Lernen beachtet werden. Im nächsten Abschnitt wurden dann die Vorteile, die CUL mit sich bringt, betrachtet. Die neuen Möglichkeiten werfen allerdings Probleme auf. Lernende sollen in Hypermediadokumenten nicht die Orientierung verlieren, Abbilder sollten so gestaltet werden, daß sie das Lernen optimal unterstützen. Diese Probleme wurden im einzelnen diskutiert, und Lösungsansätze wurden vorgestellt.

Methoden für den Entwurf von Lernsoftware konnten im Rahmen dieser Ausarbeitung nicht betrachtet werden. Es wurden auch keine existierenden Lernsysteme vorgestellt, obwohl es interessant wäre, beispielsweise ein intelligentes Tutorielles System genauer zu betrachten.

Kapitel 11

Planungskonzepte am Beispiel von MOLGEN und Anwendungen im Rahmen der Projektgruppe

11.1 Zusammenfassung

Auto: Tobias Krüger

Die vorliegende Seminararbeit zum Thema *Planungskonzepte am Beispiel von MOLGEN* im Rahmen der Projektgruppe *Virtuelle naturwissenschaftlich-technische Labore im Internet* führt zunächst in die Planung und die verschiedenen Ansätze zur Automatisierung von Planung ein. Im Anschluß werden einige Planungskonzepte vorgestellt und anhand des Planungsprogramms MOLGEN, das für die Planung von Experimenten im Genlabor eingesetzt wurde, deutlich gemacht. Zum Schluß wird diskutiert, ob diese Planungskonzepte in der Arbeit der Projektgruppe berücksichtigt werden sollten.

11.2 Einleitung

Um zum Thema Planungsprozeße hinzuführen, wird hier zunächst Planung definiert, um dann verschieden Ansätze zur Automatisierung vorzustellen.

Planung (planning) bezeichnet den Entwurfsprozeß für das Verhalten eines handlungsfähigen Wesens oder Dings. Es kann sich dabei um ein Individuum, eine Gruppe oder eine Organisation handeln. Das Ergebnis dieses Prozesses nennt man einen Plan (nach [DM95]).

Menschen stellen Pläne auf, um bestimmte Ziele zu erreichen. Es kann ein ganz alltägliches Ziel sein, wie zum Beispiel der Erwerb eines Seminarscheins. Der Student überlegt sich, wie er dieses Ziel erreichen kann. Er befindet sich in einer Ausgangssituation, besitzt ein Vorlesungsverzeichnis und sucht sich ein Seminar aus. Dort erhält er die Literatur, aus der er sich das Thema erarbeiten soll. Er hat nun einige Möglichkeiten, die ihn seinem Ziel näher bringen, wie zum Beispiel die Literatur zu lesen oder den Betreuer um Rat zu fragen. Letztendlich muß er sich einen Plan machen, wie er vorgehen will, damit

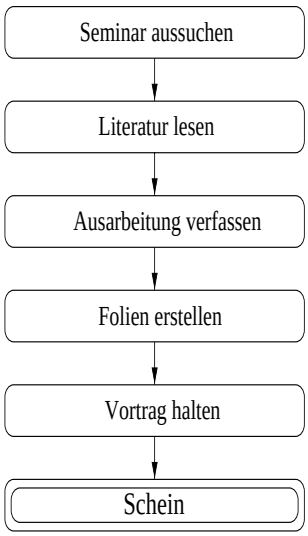


Abbildung 11.1: Beispiel eines Plans für den Erwerb eines Seminarscheins

er sein Ziel erreicht. Ein möglicher Plan könnte zum Beispiel wie in Abbildung 11.1 veranschaulicht aussehen.

Es stellt sich nun die Frage, wie sich Planung automatisieren läßt, und damit verbunden, welche Eigenschaften ein Planungsprogramm besitzen sollte. Ein Planungsprogramm sollte neue Informationen verarbeiten können, und es sollte erkennen, ob der eingeschlagene Weg zum Erfolg führt und gegebenenfalls Fehler rückgängig machen können (nach [Ste81a]).

Zu Planungsprogrammen gibt es verschiedene Ansätze.

Der *klassische Ansatz* (classical Artificial Intelligence planning) wurde mit dem General Problem Solver (GPS) realisiert. Dieses Programm war prinzipiell dazu in der Lage jedes Suchproblem (search problem) zu lösen. Suchprobleme stellen allerdings nur einen kleinen Ausschnitt der Planungsprobleme dar.

Klassische Planung geht von den folgenden Annahmen aus:

- Das Ergebnis jeder Handlung ist vorhersagbar.
- Der Zweck eines Plans ist das Erreichen eines vorher beschriebenen Ziels.
- Pläne sind Handlungen in einer bestimmten Reihenfolge.

Diese Sichtweise eignet sich insbesondere für Ablaufplanung, bei der Handlungen nur sequentiell abgearbeitet werden müssen, ist aber unvollständig. Beispielsweise wird bei der Planung der Handlung eines Roboters nicht in Betracht gezogen, daß sich die Umwelt verändern kann. Weder das Verhalten der Umwelt läßt sich immer eindeutig vorhersagen, noch eine Situation als solche ist vollständig bekannt.

Daraus hat sich die *dynamische Planung* (dynamical world planner) entwickelt. Um der Dynamik der Umwelt Rechnung zu tragen, wurden die Reize (stimulus) direkt mit der Reaktion (response) gekoppelt. Zugleich ergab sich das Problem der Limitierung der Zeit. Fließbandroboter haben zum Beispiel nur

eine begrenzte Zeit um das zu bearbeitende Objekt zu erkennen und dafür einen Plan zu entwickeln. Also wird nach einer bestimmten Zeit t, die dem Roboter zur Verfügung steht, der bisher beste Plan verwendet, unabhängig davon, ob es vielleicht noch bessere Pläne gibt.

Ein weitere Möglichkeit, Planung effizient zu automatisieren, ist, die Programme gemäß dem Bereich, für den die Planung bestimmt ist, zu spezialisieren, diese Planungsprogramme nennt man *special-purpose planner*. Das bedeutet, es werden Beschränkungen und Eigenschaften des Bereichs direkt in das Planungsprogramm übernommen, um so die Effizienz zu erhöhen. Der Nachteil ist, daß sich die so realisierten Programme nicht auf andere Bereiche (Domänen) übertragen lassen.

Bis jetzt wurde noch kein Planungsprogramm, geschweigen denn ein Planungskonzept, also eine systematische Herangehensweise an Planung, entwickelt, das für alle Probleme, bei denen eine Planung erforderlich ist, gleichermaßen geeignet wäre.

11.3 Planungskonzepte

In diesem Abschnitt werden als erstes intuitive Herangehensweisen bezüglich Planung beschrieben, die auch von Menschen benutzt werden, um dann in konkretere bzw. speziellere Konzepte, die in Planungsprogrammen verwendet werden, einzuführen.

11.3.1 System und Teilsystem

Komplexe Systeme betrachtet man am besten, nicht indem man es als Ganzes wahrnimmt, sondern vielmehr indem man das System in geeignete Teilsysteme zerlegt, die weniger komplex sind und die schon allein durch ihre geringere Größe besser zu verstehen sind. Die Beziehungen zwischen den Teilsystemen werden dabei häufig zunächst vernachlässigt, denn wenn man ständig die Beziehungen und Abhängigkeiten mit in Betracht ziehen müßte, würde sich kein Vorteil ergeben.

Ebenso verhält es sich mit dem Entwurfsprozeß komplexer Systeme. Dort teilt man die Systeme bereits im Planungsprozeß, also auf einer abstrakten Ebene, auf, entwirft jeweils nur Teile des Gesamtsystems. Das hat u.a. folgende Gründe:

- Die Komplexität wird scheinbar verringert.
- Die Aufteilung kann vor der Spezifizierung der einzelnen Teilsysteme erfolgen, da die Einzelheiten für die betrachtete, abstrakte Ebene irrelevant sind.
- Die Spezifizierung der Teilsysteme kann jeweils von Fachleuten vorgenommen werden.

Es muß hierbei beachtet werden, daß die Teilsysteme voneinander abhängig sind. Man muß also geeignete Aufteilungen wählen, um die Abhängigkeiten zu minimieren und sie so doch einbeziehen zu können.

Am Beispiel der Seminararbeit läßt sich das veranschaulichen. Man entwirft einen Plan, der noch relativ abstrakte Schritte enthält. So hat man noch keine Kenntnis über die genaue Art, wie der Vortrag zu halten ist oder wie die Ausarbeitung genau aussehen wird oder muß. In diesem Beispiel gibt es zeitliche Abhängigkeiten, denn wenn man noch keine Literatur gelesen hat, kann man nicht die Folien erstellen oder den Vortrag halten.

11.3.2 Hierarchische Planung

Hierarchische Planungsprogramme gehen wie im Abschnitt 11.3.1 beschrieben vor. Sie teilen das Gesamtproblem in Teilprobleme auf, zusätzlich besitzen sie die Fähigkeit der Abstraktion. Sie stellen also einen Plan zunächst auf einer abstrakten Ebene auf und schieben die Betrachtung der Details auf. Danach versuchen sie Schritt für Schritt die abstrahierten Elemente zu spezifizieren und zu instantiieren (refinement), um so zu einem konkreten Plan zu kommen.

Darin unterscheiden sie sich auch von nicht-hierarchischen Planungsprogrammen. Letztere versuchen sofort den Plan zu konkretisieren, d.h. wenn ihnen eine Information fehlt, versuchen sie, diese sofort zu bestimmen, und warten nicht, bis vielleicht die Spezifizierung eines anderen Teils zur richtigen Spezifizierung der fehlenden Information führt.

Ein Beispiel für hierarchische Planung wäre analog zum Beispiel aus dem vorhergehenden Abschnitt. Zunächst wird der abstrakte Vorgehensplan aufgestellt, ohne daß die zu lesende Literatur schon im einzelnen bekannt wäre. Es handelt sich also noch um einen universellen Plan, der prinzipiell auf jedes Seminar anzuwenden wäre. Erst später wird dann die Literatur spezifiziert.

11.3.3 Constraints

Wie im Abschnitt 11.3.1 über Systeme und ihre Zerlegung in Teilsysteme angesprochen, sind die Teilsysteme im Normalfall voneinander abhängig, so daß kein vollständig getrennter Entwurf der Teilsysteme bzw. Lösung der Teilprobleme erfolgen kann. Dies bedeutet, daß man in hierarchischen Planungsprogrammen ein Werkzeug verwenden muß, das die Beziehungen zwischen den abstrakten Teilproblemen beschreibt. Man formuliert dazu *constraints* (Einschränkung, Zwang). Sie geben die Relationen verschiedener Variablen des Plans wieder.

Constraints werden insbesondere in MOLGEN verwendet. Sie sind bereits ein spezielleres Planungskonzept, werden in diesem Kapitel aber noch allgemein abgehandelt. Im Kapitel 11.4 wird das Konzept veranschaulicht.

11.3.3.1 Bedeutungen von Constraints

Zum einen kann man constraints als *Ausschlußregeln* (elimination rules) betrachten. Sie stellen eine Bedingung für die Variablen dar und schränken so die Menge der erlaubten Belegungen ein.

Zum Beispiel schränkt die Aussage „Das Seminar soll nicht aus dem Bereich Technische Informatik stammen“ die Menge der erlaubten Belegungen für die Variable Seminar bereits ein. Man hat nun nur noch sechs statt acht mögliche Seminare.

Zum anderen bieten constraints eine *Teilbeschreibung der Variablen* (partial description). Im least-commitment-Ansatz werden Entscheidungen über tatsächliche Belegungen möglichst lange aufgeschoben. Constraints bieten hier die Möglichkeit, Variablen noch nicht vollständig zu beschreiben, d.h. zu instantiieren, sondern zunächst nur teilweise zu beschreiben. Der Plan wird also verfeinert, kommt der Lösung näher, ohne daß Variablen tatsächlich belegt werden müssen.

Die Beschreibung „Das Seminar soll mit Multimedia zu tun haben“ verfeinert den Plan dadurch, daß die Variable Seminar schon sehr genau beschrieben ist, aber trotzdem noch nicht instantiiert ist.

Außerdem kann man constraints als *Kommunikationsmittel* (communication medium) einsetzen, um Beziehungen der Teilsysteme darzustellen. Da eine Variable häufig in mehreren Schritten des Plans benutzt wird, kann man constraints von einem Teil des Plans auf einen anderen Teil übertragen.

Die Pflicht, einen Vortrag zu halten, kann zum Beispiel bedeuten, daß man sich darauf gesondert vorbereiten muß und vielleicht noch einmal ein Buch zum diesem Thema anschaut, um sich Anregungen zu holen. Die Bedingung, die sich aus dem Vortrag ergibt, wird also auf den Schritt des Lesens von Literatur übertragen, von einem Planschritt auf einen anderen. Hierzu im Abschnitt 11.3.3.2 mehr.

11.3.3.2 Operationen auf Constraints

Es gibt drei wichtige Operationen auf Constraints, die einem Programm, das constraints benutzt, zur Verfügung stehen sollten:

- Die *Formulierung von Constraints* (constraint formulation) ist notwendig, wenn sich im Zuge der Spezifizierung des Plans neue Bedingungen ergeben.
- Die *Übertragung von Constraints* (constraint propagation) sollte über den gesamten Plan hin möglich sein, d.h., wird eine neue Bedingung in einem Teil des Plans formuliert, muß sie auch auf einen anderen Teil des Plans übertragbar sein, wenn dort die gleiche Variable benutzt wird.
- Außerdem muß es möglich sein, bei der Belegung von Variablen mit Werten die *Constraints zu berücksichtigen*, also Belegungen zu finden, die den Bedingungen genügen (constraint satisfaction).

11.3.4 Meta-Planning

Viele Ziele, Handlungen und Bedingungen sind nicht direkt auf der Ebene der Domäne, sondern sozusagen auf einer Meta-Ebene angesiedelt und gelten für den Planungsprozeß bzw. den Plan selbst. Zum Beispiel muß man die Seminararbeit innerhalb einer bestimmten Zeit fertiggestellt haben und daher auch den Planungsprozeß zeitlich begrenzen. Ein Ziel auf einer höheren Ebene wäre zum Beispiel auch die Anzahl der Schritte zur Lösung des Problems.

Um diesem Problem zu begegnen, teilt man das Planungsprogramm in verschiedene Ebenen ein. Man kann eine Domänenebene unterscheiden und eine Planungsebene, wobei die Planungsebene selbst wieder aus mehreren Ebenen zusammengesetzt sein kann. In der Planungsebene können die Meta-Ziele verarbeitet und berücksichtigt werden. Hier kann unter anderem der Lösungsweg kontrolliert werden.

MOLGEN besitzt zwei Planungsebenen, die im nächsten Abschnitt genauer erklärt werden.

11.4 Planungskonzepte am Beispiel von MOLGEN

Die Planungskonzepte werden anhand eines (vereinfachten) Beispiels für einen Planungsentwurf deutlicher.

11.4.1 Planungsaufgabe

Das Programm MOLGEN wurde im Rahmen eines Projekts zur Erforschung der heuristischen Programmierung (heuristic programming) im Department of Computer Science an der Stanford University, Kalifornien, entwickelt [Ste81b]. Es soll im Bereich der Molekularbiologie die Planung eines Experiments unterstützen. Gegeben sind Ziele der Synthese und Analyse, Wissen über die Laborausrüstung und -techniken, über deren Eigenschaften und Verwendung.

11.4.2 Aufbau von MOLGEN

MOLGEN ist gemäß dem in Abschnitt 11.3.4 beschriebenen Meta-Planning-Konzept aufgebaut. Es besitzt drei Ebenen (spaces), die Ebene der Domäne (*laboratory space*), die Planentwurfsebene (*design space*) und die Strategieebene (*strategy space*), die in Abbildung 11.2 veranschaulicht sind.

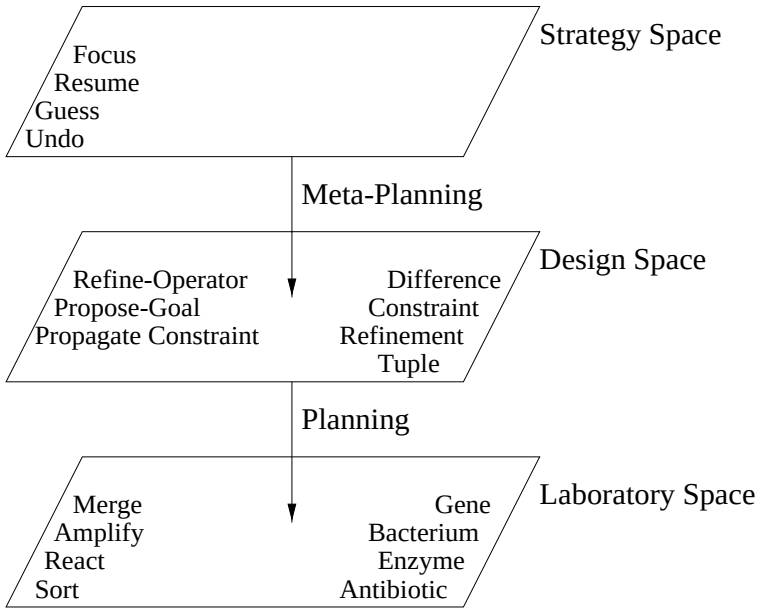


Abbildung 11.2: MOLGENs drei Ebenen - laboratory, design und strategy space

11.4.2.1 Wissensbasis

MOLGEN besitzt eine wenn auch beschränkte Wissensbasis. Es hat zahlreiche Materialien zur Verfügung, sowie gewisse Standardobjekte, wie zum Beispiel ein prototypisches Bakterium. Die verfügbaren Operationen sind im Laboratory space abgebildet.

11.4.2.2 Laboratory space

Die unterste Ebene hält die möglichen Schritte im Labor vor (laboratory space). Diese sind noch einmal gruppiert und mit Oberbegriffen versehen.

Folgenden Schritte sind möglich:

- Merge: Objekte zusammenfügen
- Amplify: Objekte vervielfachen
- React: Objekte bzw. deren Eigenschaften verändern
- Sort: Objekte sortieren

Die aufgezählten Schritte sind noch abstrakt, die genauen Operationen lassen sich der Abbildung 11.3 entnehmen. Eine Spezifizierung des Sort-Operators ist zum Beispiel Screen. Der Sort Operator beschreibt, welche Handlung durchzuführen ist, die Spezifizierung ist das Screening, d.h. hier werden bestimmte unerwünschte Bakterien mit Hilfe eines Antibiotikums abgetötet.

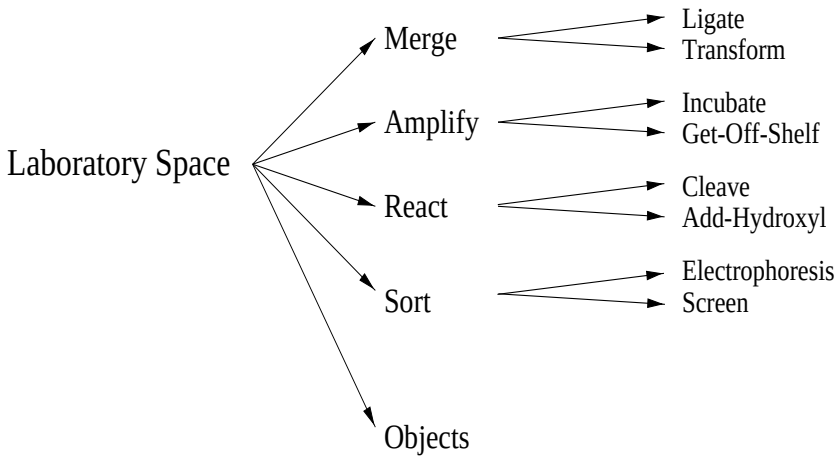


Abbildung 11.3: Laboratory Space - abstrakte und spezifische Operatoren

11.4.2.3 Design space

Im Entwurfsbereich werden Operatoren vorgehalten, die die Operatoren im laboratory space kontrollieren, sogenannte *planning operators*. Es kann zwischen folgenden Operatoren unterschieden werden:

Die *Vergleichsoperatoren* vergleichen Objekte miteinander (Find-unusual-Features), zum Beispiel tatsächliche vorhandene (als Ausgangsbasis) und gewünschte, und berechnen ihren Unterschied, um dann einen Operator des laboraty space vorzuschlagen, der die Unterschiede verringert, oder sie überprüfen, ob durch einen Schritt tatsächlich das gewünschte Ziel erreicht wird (Check-Prediction).

Die *Temporal-extension-Operatoren* dehnen einen Plan zeitlich vorwärts oder rückwärts aus. Sie schlagen Operatoren oder Ziele vor (Propose-Operator bzw. Propose-Goal) oder berechnen das Ziel eines vorgeschlagenen Schrittes (Predict-Results).

Die *Spezifikations-Operatoren* (specialization operators) fügen zu einem noch abstrakten Plan Details hinzu. Sie können Operatoren und Objekte spezifizieren (Refine-Operator und Refine-Object) und sie können Constraints hinzufügen, die sich aus bereits bestehenden Constraints ergeben (Propagate-Contraint).

11.4.2.4 Strategy Space

Der strategy space hat die Aufgabe, die Designschritte zu kontrollieren. Er wählt Schritte des Designbereichs aus und führt diese aus. Die Operatoren nennt man *meta-planning* oder *strategy operators*.

Der Strategiebereich in MOLGEN ist durch zwei Ansätze geprägt, der *least-commitment approach* und der *heuristic approach*. Generell befindet sich MOLGEN im least-commitment cycle. Hier erweitert es den Plan, indem es neue Designschritte entwirft (Operator Focus) oder alte aufgeschobene Schritte weiter spezifiziert (Operator Resume).

Erst wenn dieser Prozeß stehen geblieben ist, weil keine neuen Schritte hinzugefügt werden können und alle bestehenden Schritte keine eindeutige Verfeinerung mehr zulassen, d.h immer noch mehrere Belegungen für die Variablen möglich sind (under-constrained), erst dann begibt sich MOLGEN in den heuristic cycle, in dem es die Belegung einer Variablen rät (guess). Danach begibt sich MOLGEN zurück in den ersten Zyklus. Nun kann MOLGEN entweder den Plan weiter detaillieren, oder aber es stellt fest, daß sich durch die Belegung ein Widerspruch gebildet hat (over-constrained). In diesem Fall begibt sich MOLGEN wieder in den heuristischen Zyklus und macht die Wahl rückgängig (undo). Die beiden Zyklen sind in Abbildung 11.4 dargestellt.

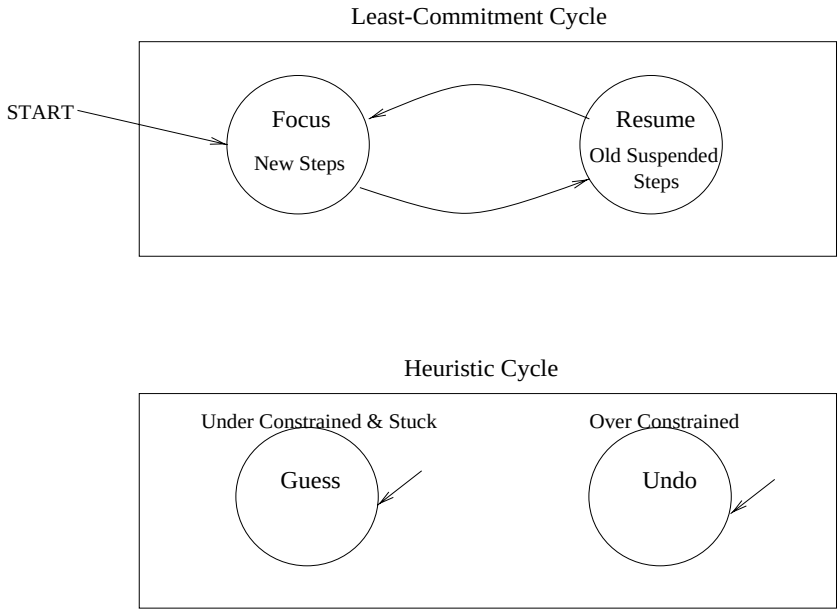


Abbildung 11.4: Least-commitment und heuristic cycle im strategy space von MOLGEN

11.4.3 Das Experiment

Eines der Ziele von Genexperimenten ist, Bakterien für die Produktion eines Proteins zu nutzen. Im vorgestellten Experiment (nach [Ste81b]) soll ein Bakterium modifiziert werden, das ein bestimmtes Gen, nämlich Ratten-Insulin enthält. Dazu benutzt man einen Vektor, der die Überführung des Gens in das Bakterium ermöglicht. Danach wird das Bakterium vervielfältigt, um so eine Art Insulin-Fabrik zu erhalten.

11.4.3.1 Erste Schritte

Man gibt dem Programm nun vor, daß es einen Plan für die Herstellung eines unbestimmten Bakteriums mit einem unbestimmten Vektor, der das Gen Ratten-Insulin enthält, entwickeln soll.

MOLGEN beginnt mit dem Operator *Focus*, durch den einen Designschritt aufgestellt werden soll. Als Designschritt wird *Find-Unusual-Features* verwendet. MOLGEN versucht also einen Unterschied oder eine Besonderheit des Ziels zu finden und vergleicht das gewünschte Bakterium mit dem prototypischen Bakterium seiner Wissensbasis. Die Besonderheit, die MOLGEN findet, besteht darin, daß das Bakterium einen besonderen Vektor mit einem bestimmen Gen, nämlich dem Ratten-Insulin-Gen enthält. Dieser Unterschied wird nun als Eingabeparameter dem Operator *Propose-Operators* übergeben, der einen Operator finden soll, um den Unterschied zu verringern. *Propose-Operators* wiederum findet den abstrakten Merge-Operator im Laboratory space, sagt jedoch noch nichts über die vorhergehenden Schritte oder über die zu vermischenden Objekte aus.

Eine schematische Darstellung findet sich in Abbildung 11.5.

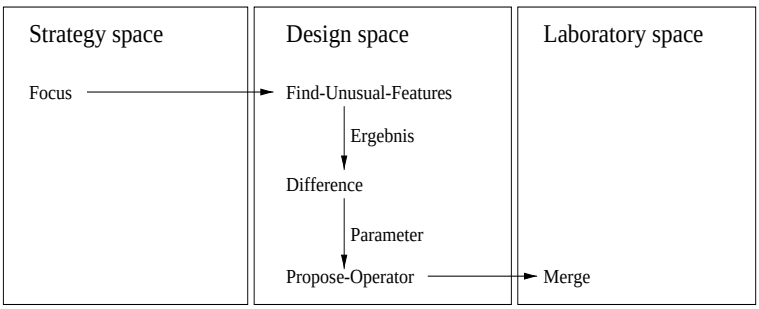


Abbildung 11.5: Schema von MOLGENs Schritten, um den ersten Operator zu finden

Die Folgeschritte von MOLGEN, die hier der Kürze wegen nicht aufgeführt sind, führen zu einem genaueren Plan, wie Abbildung 11.6 veranschaulicht.

11.4.3.2 Verfeinerung

Im weiteren Verlauf der Planung vollzieht MOLGEN folgenden Schritt: Es ersetzt den abstrakten Merge-Operator der letzten Stufe durch den konkreten *Transform-Operator*. Hier kommt eine Eigenschaft von MOLGEN zur Anwendung, nämlich die *Formulierung von Constraints*. Der Transform-Operator verlangt, daß die beiden Objekte, die verschmolzen werden sollen, biologisch kompatibel sind. Dieses ist also die erste Bedingung des Plans und schränkt somit die Lösungsmenge ein (vergleiche Abbildung 11.7).

Gemäß dem least-commitment-Ansatz instantiiert MOLGEN die beiden Variablen *Bakterium* und *Vektor* nicht sofort, denn bei der weiteren Verfeinerung des Plans könnten neue Constraints entdeckt werden, zu denen die Variablenbelegung im Widerspruch stehen könnte. Außerdem gibt es noch sehr viele Möglichkeiten, so daß die Chance, eine tatsächlich gültige Belegung zu finden, gerade zu Beginn der Planung nicht sehr groß ist. MOLGEN behält diese Bedingung nur bei.

Mit Hilfe des *Predict-Result-Operators* kann MOLGEN das Ergebnis des *Transform*-Schrittes voraussagen. Dabei entnimmt MOLGEN seiner Wissensbasis, daß durch die Umwandlung nicht alle Bakterien mit dem gewünschten Vektor versehen werden können. Es handelt sich sozusagen um eine aus

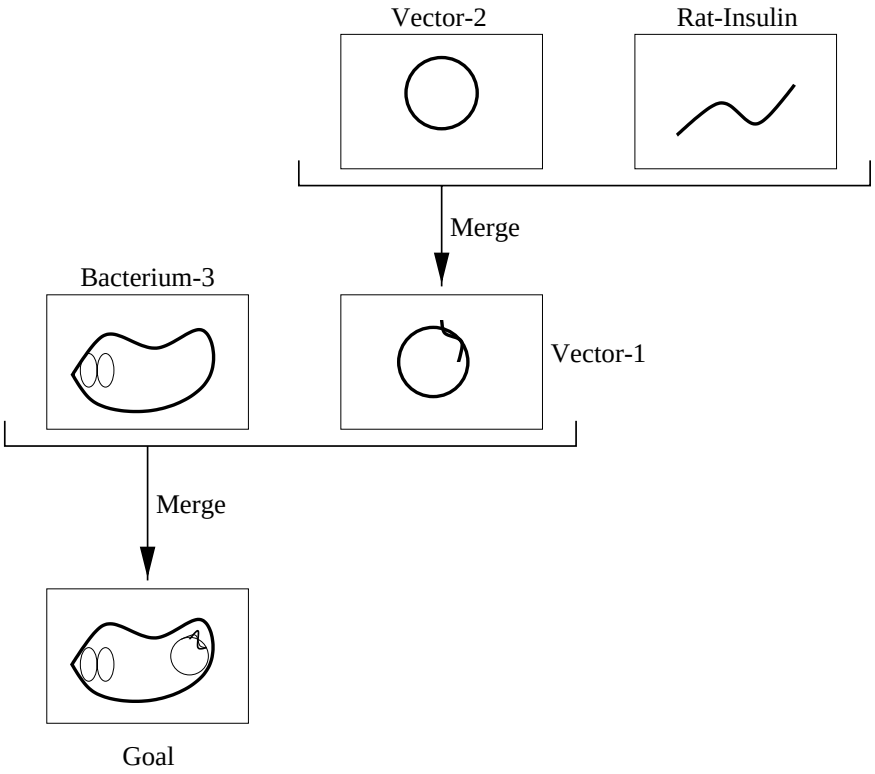


Abbildung 11.6: Zwischenergebnis MOLGENs - ein noch abstrakter Plan

der Chemie bekannte Gleichgewichtsreaktion. Es bleiben immer einige Bakterien übrig, die unverändert sind (vergleiche Abbildung 11.8). Das entspricht allerdings nicht dem Endziel, bei dem nur Bakterien mit dem speziellen Vektor verlangt wurden.

Um diesen Unterschied zu verringern, muß MOLGEN einen Operator finden, der die unerwünschten Bakterien herausfiltert. Es schlägt das abstrakte *Sort* vor. Danach wird dieses in das konkrete *Screen* umgewandelt, bei dem mit Hilfe eines Antibiotikums bestimmte Bakterien herausgefiltert werden. Dieser Operator führt zu einer Formulierung zweier neuer Bedingungen. Zum einen muß das Bakterium mit dem Vektor gegen das Antibiotikum resistent sein, zum anderen darf das Bakterium ohne den speziellen Vektor nicht resistent sein.

An dieser Stelle tritt eine andere Eigenschaft bezüglich Constraints hervor: *Contraint propagation*. Da die Variable Bakterium auch in früheren Schritten schon verwendet wird, überträgt MOLGEN die erst am Schluß des Versuchs geforderte Eigenschaft, auf einen früheren Zeitpunkt. In diesem Fall wird die Bedingung auf den Vektor, der ganz am Anfang dazu bestimmt ist, das Ratten-Insulin zu tragen, übertragen. Er muß nun ein Gen für die Resistenz des am Schluß verwendeten Antibiotikums tragen. Die Übetragug der constraints ist in der Abbildung 11.9 schematisch dargestellt.

Die dritte Eigenschaft bezüglich Constraints, *constraint satisfaction*, läßt sich am Beispiel des Bakteriums und des Vektors, die laut der ersten Bedingung biologisch kompatibel sein müssen, zeigen. MOLGEN durchsucht einfach seine Wissensbasis nach Objekten, die der Bedingung genügen. Es berücksichtigt dabei alle Constraints, die mit der jeweiligen Variable in Beziehung stehen.

Mit diesen möglichen Operationen kann MOLGEN den Plan zu Ende führen. Es führt neue Cons-

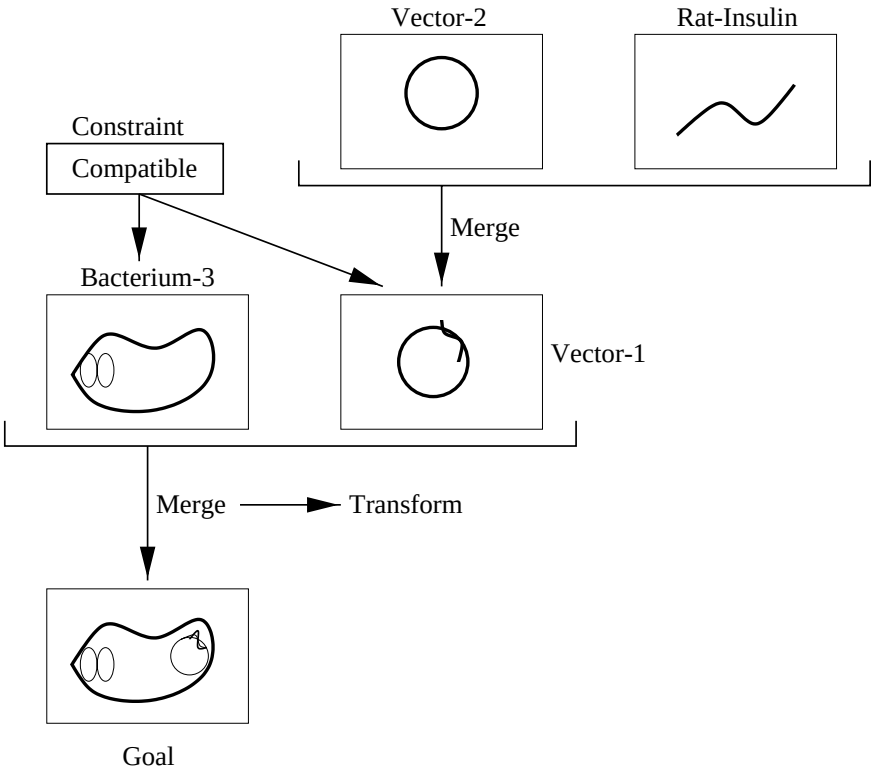


Abbildung 11.7: Verfeinerung des Plans - die erste Bedingung (constraint)

traints, bei Bedarf auch neue Variablen ein, es überträgt neue Bedingungen, es sucht nach passenden Werten für die Variablen. Wenn der Plan an einer Stelle stockt, kann es weitere Schritte erraten und auch wieder rückgängig machen, wenn sie nicht zum Erfolg führen.

Die Effektivität wird an der Zahl der von MOLGEN während des Planungsentwurfs betrachteten Belegungen deutlich. Insgesamt gibt es am Anfang 3456 Möglichkeiten (Produkt der Belegungen für die verschiedenen Variablen). MOLGEN betrachtet aber während der Planungsphase maximal 21 verschiedene Belegungsmöglichkeiten. Es weiß also geschickt den Ballast zu vermeiden und betrachtet nur die unbedingt notwendigen Variablenbelegungen.

11.5 Anwendungen im Rahmen der Projektgruppe

> „Das Ziel der Projektgruppe *Virtuelle naturwissenschaftlich-technische Labore im Internet* ist die Konzeption und Realisierung eines Werkzeugkastens für die Entwicklung virtueller naturwissenschaftlich-technischer Labore im Internet. Die entstehenden Labore sollen zur elektronischen Vermittlung theoretischer und praktischer naturwissenschaftlicher Grundlagen der jeweiligen Naturwissenschaft und zur Unterstützung des naturwissenschaftlichen Experimentierens dienen. Dazu sollen zunächst die charakteristischen Merkmale realer Labore analysiert und modelliert werden. “(nach [DB97])

Das Ziel der Projektgruppe ist demnach nicht die Entwicklung eines Programms, das naturwissenschaftliche Experimente plant, so wie MOLGEN das tut. Vielmehr soll es mit Hilfe des Werkzeugkastens

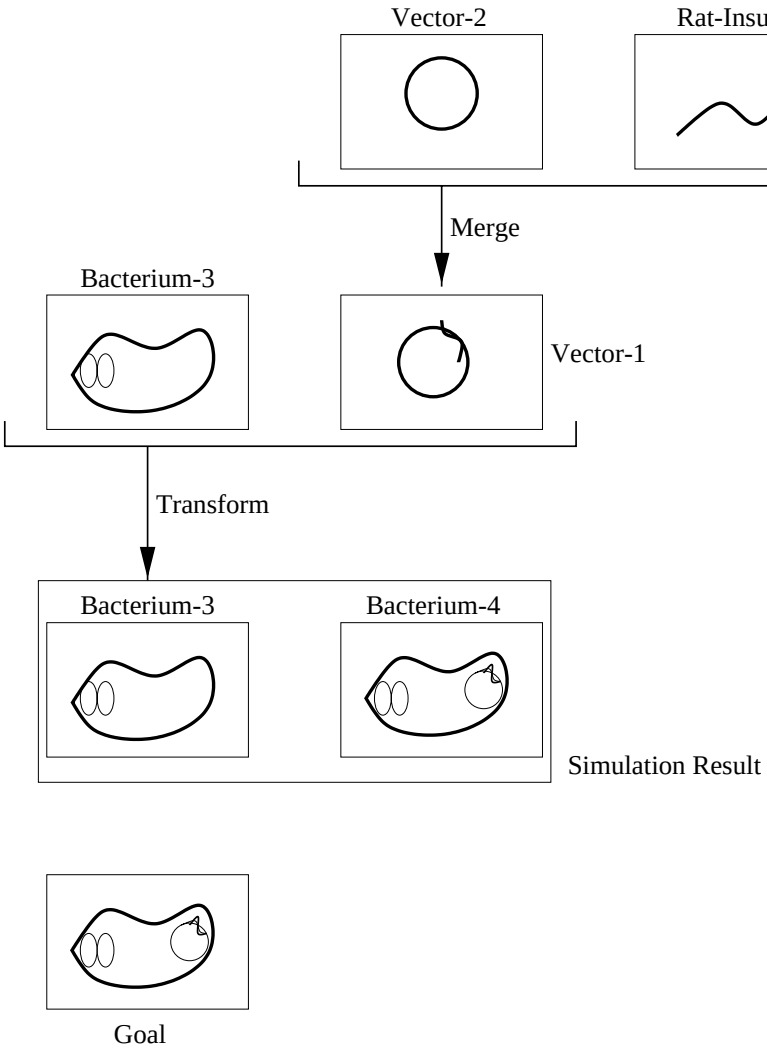


Abbildung 11.8: Das Simulationsergebnis ergibt eine Abweichung zum Ziel

möglich sein, eine Umgebung zu schaffen, die Experimente, Laborgeräte und Materialien simuliert. Deshalb wird im folgenden zunächst die Simulationskomponente betrachtet.

11.5.1 Simulationskomponente

Auf den ersten Blick scheinen dies völlig verschiedene Programme zu sein. Beim näheren Betrachten stellt man jedoch fest, daß beide Programme viele Gemeinsamkeiten aufweisen.

Sie simulieren beide ein Labor und machen die Planung eines Experimentes möglich. Das bedeutet, daß MOLGEN, wie beschrieben, und auch die Projektgruppen-Simulationsumgebung (PGS) eine Wissensbasis besitzen müssen, die das Labor, die beinhalteten Geräte und Materialien, sowie ihre Eigenschaften und Funktionen, beschreibt.

Es muß in beiden Fällen eine Überprüfung der Gültigkeit des Lösungsweges stattfinden.

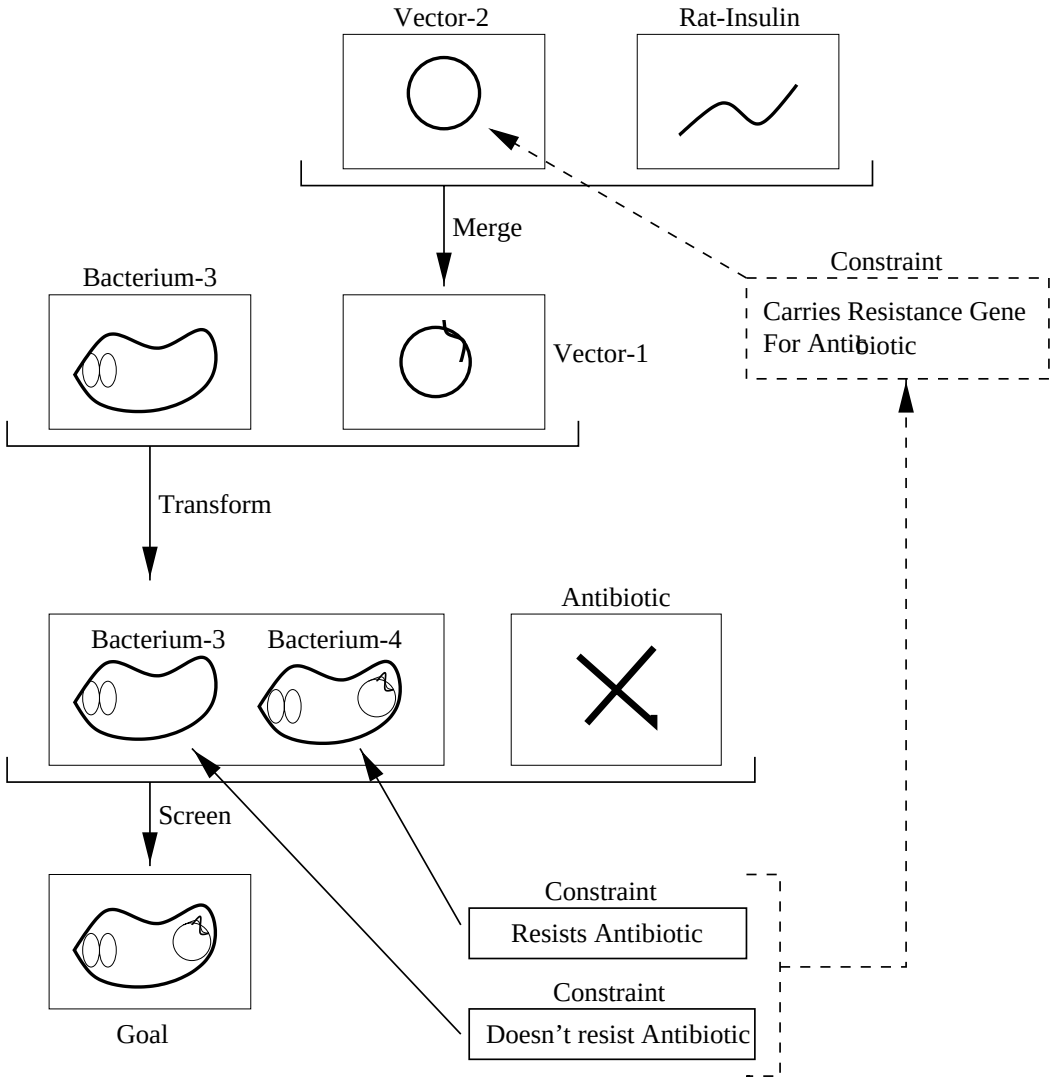


Abbildung 11.9: Propagating contraints - schematische Darstellung

Im Falle MOLGENs geschieht das inhärent. Die Operatoren der Design-Ebene schlagen nur Operatoren vor, die auch tatsächlich verwendet werden können. Die Regeln werden autmatisch befolgt, da kein Nutzer eingreifen kann. Auch wenn geraten wird, kontrolliert MOLGEN, ob sich im Plan nicht Widersprüche ergeben und revidiert gegebenenfalls seine Wahl.

Im Falle der von der Projektgruppe zu erstellenden Simulationsumgebung ist das etwas schwieriger. Es gibt mehrere Ansätze, die sich in den Möglichkeiten unterscheiden, die man dem Anwender bietet. Die ist in Abbildung 11.10 dargestellt:

Der einfachste Weg wäre sicherlich, dem Benutzer den Lösungsweg vorzugeben. Man kann dies natürlich mehr oder weniger aufwendig gestalten, aber prinzipiell müßte der Anwender nur den nächsten Schritt abrufen und anschauen. Eigene Gedanken zum Lösungsweg kann er sich, muß er sich aber nicht machen, da diese dem Programm nicht mitgeteilt werden können. Es existiert somit keine Schnittstelle. Es ist bei diesem Ansatz auch unwichtig, ob man dem Anwender zunächst einen abstrakten Plan oder den Detailplan zeigt. Der abstrakte Plan wäre allerdings pädagogisch sinnvoller, da die Vorgehensweise

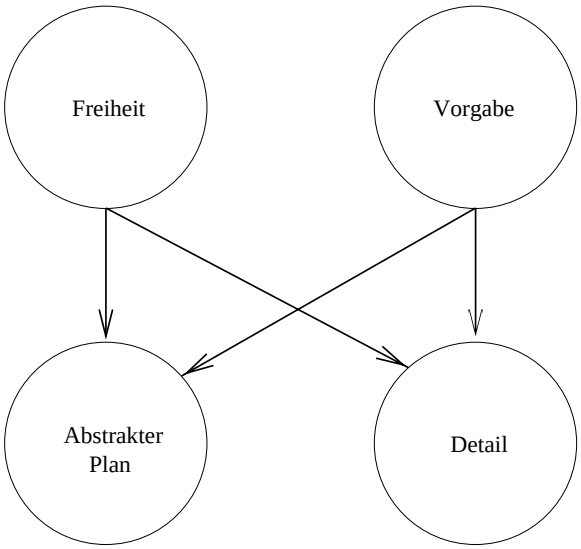


Abbildung 11.10: Ansätze für die Simulation eines Labors - Zwischen Freiheit und Vorgabe, abstrakter Planung und Details

der menschlichen angepaßter wäre (vergleiche das Einleitungskapitel).

Räumt man dem Anwender größere Freiheiten ein, könnte man ihm eine virtuelle Laborumgebung präsentieren, die nur konkrete Objekte enthält. Der Anwender müßte sich selbst einen geeignete abstrakten Plan überlegen, der von dem Programm nicht überprüft würde. Der Anwender müßte von Beginn an mit konkreten Objekte arbeiten. Es bliebe seinem Verstand überlassen, ob er den abstrakten Plan umsetzen kann. Das Programm kann ihn dabei entweder völlig alleine lassen und nur das erreichte Ergebnis mit dem gewünschten Ziel vergleichen. Es konnte dem Anwender aber in einer Stocksituation auch eine Hilfe anbieten. Bei dieser wären verschiedene Ausprägungen denkbar. Die Möglichkeiten reichen von einer reinen Lösungsvorgabe bis hin zu einer kontextsensitiven Hilfe, die dem Anwender Ratschläge und Tips zur Lösungsfindung gibt. Eine kontextsensitive Hilfe ist auch als Planungssystem vorstellbar, das versucht, von der Situation, in der der Anwender Hilfe ersucht, einen Lösungsweg zu finden und daraufhin bestimmte Tips gibt.

Wenn man dem Anwender die Möglichkeit bietet, auch seinen abstrakten Plan in das Programm einzugeben, stellt sich nicht nur die Frage nach der Repräsentation. Ebenso ist zu überlegen, ob und ggf. wie man diesen abstrakten Plan überprüft. Man müßte auch in diesem Fall ein Planungsprogramm wie MOLGEN mitlaufen lassen, daß den abstrakten Plan versucht zu verfeinern und durch einen Vergleich mit einem eigenen abstrakten Plan auch Hinweise gibt oder Korrekturen vornimmt. Das ist allerdings ein sehr großer Aufwand.

Der Ansatz, dem Anwender eine größere Freiheit zu lassen, ihn aber nur mit den konkreten Objekten zu konfrontieren und ihm selbst die abstrakte Planung zu überlassen, scheint am effektivsten. Man kann dem Anwender, sofern ein vorgegebenes Ziel zu erreichen ist, eine kontextabhängige Hilfe bieten. Diese Hilfe läßt sich mehr oder weniger aufwendig gestalten. Sie kann sich dem Anwender anpassen und die Hilfe auf ihn und seine Bedürfnisse abstimmen, oder einfach eine Standardhilfe, die, wenn der Anwender es will, ihm einen Lösungsschritt vorgibt.

Ein weitere Vorteil ist, daß der Anwender auch ohne Ziel experimentieren kann. Er kann mit den

Materialien arbeiten, um zu erfahren, welche Eigenschaften sie haben. Er kann bestimmte Laborgeräte ausprobieren und ihre Funktionen erkunden.

11.5.2 Autorensystemkomponente

Nun stellt sich die Frage, welche Anwendungen man sich in einer Autorensystemkomponente vorstellen könnte. Es wäre sinnvoll, dem Autor die Möglichkeit zu bieten, eine neue Wissensbasis zu definieren. Es müßte ihm möglich sein, die Wissensbasis für ein neues Labor herzustellen. Wenn man ein Planungsprogramm implementiert hat, das auf jeder Wissensbasis operieren kann, die von der Komponente generiert wird, so müßte der Autor darüberhinaus nur noch gewisse Ziele von Experimenten angeben. Die Simulationskomponente könnte dann sowohl die Gültigkeit des Zieles überprüfen, als auch die Gültigkeit des Lösungsweges, die der Anwender (Nutzer) einschlägt. Das Programm könnte dem Anwender dann eine kontextsensitive Hilfe bieten. Wenn man ohne Planungssystem arbeitet, müßte der Autor zum Beispiel den Lösungsweg angeben, je nachdem welche Art von Hilfe man benutzen möchte.

11.6 Abschließende Anmerkungen

Die Planungsprogramme lassen sich bis dato nur für spezielle Bereich nutzen. Sie können Suchprobleme gut lösen, bei denen es auf Rechenstärke ankommt, sie können inzwischen auch die Umwelt einbeziehen, aber sie sind doch nur für eingeschränkte Bereiche zu gebrauchen. Auch MOLGEN ist auf das Gebiet der Genexperimentplanung fixiert. Es bietet in diesem Bereich keine große Unterstützung, war aber auch nicht dafür vorgesehen. So bleiben die Forscher bis auf weiteres auf ihre Erfahrung und Intuition angewiesen.

Die Planungskonzepte lassen sich zum Teil auch bei der Simulation eines Labors verwenden. Inwieweit man sich hier dem Menschen anpaßt und intuitive Lösungsstrategien unterstützt, bleibt dem Entwickler überlassen. Es scheint jedoch aus der Sicht der Lehr-/Lernsysteme vernünftig, nicht die bloße Vorgabe von Wissen anzustreben. Das ist vielleicht bei der Vorstellung von Laborgeräten und Materialien angebracht. Beim Experiment den Lösungsweg vorzugeben und keine Interaktion zuzulassen, würde den Lernerfolg jedoch mindern. Hier sollte dem Anwender alle Freiheiten gelassen werden und mit einer möglichst guten kontextsensitiven Hinweisen geholfen werden. Gegen ein Planungsprogramm spricht der sehr große Aufwand, den die Implementierung mit sich bringt. Dies ist meiner Meinung nach im Rahmen der Projektgruppe nicht möglich, sollte aber noch einmal überprüft werden.

Literaturverzeichnis

[adUO96] Projektgruppe an der Uni Oldenburg. Softwareergonomische aspekte bei gestaltung www-basierter lernsoftware. http://www-cg-hci.informatik.uni-oldenburg.de/~pgse96/, 1996.

[AK98] Stephan Ehrmann Andreas Kiel. Medienfusion. *c't*, (7):128, 1998.

[app] http://www.quicktime.apple.com.

[App97] Apple Computer. *QuickTime VR Authoring Studio User Manual*, erste edition, 1997.

[Bal98] Helmut Balzert. *Software-Technik*. Spektrum Akad. Verlag, erste edition, 1998.

[Bey93] M. Beyer. *BrainLand: Mind Mapping in Aktion*. Junfermannsche Verlagsbuchhandlung, 1993.

[Bol95] D Boles. *Elektronisches Publizieren-Autorensysteme und Arbeitsumgebungen für Autoren*. Universität Oldenburg, Fachbereich Informatik, 1995.

[Bol97a] D. Boles. *Erstellung multimedialer Dokumente und Anwendungen*. Universität Oldenburg, Fachbereich Informatik, 1997.

[Bol97b] D. Boles. *Skript zur Vorlesung „Multimedia-Systeme“*. Universität Oldenburg, Fachbereich Informatik, 1997.

[Bol98] Boles/Schlattmann. *Multimedia-Autorensysteme: Graphisch-interaktive Werkzeuge zur Erstellung multimedialer Anwendungen.* LOGIN, Heft 1, 1998, 1998.

[Boo94] Grady Booch. *Object Oriented Analysis and Design with Applications*. Benjamin/Cummings Publishing Company, 1994.

[Bor95] Uwe Borghoff. *Rechnergestützte Gruppenarbeit: Eine Einführung in verteilte Anwendungen*. Springer-Verlag, Berlin Heidelberg, 1995.

[Bur93] Jeff Burger. *The Desktop Multimedia Bible*. Addison-Wesley, 1993.

[Bur94] Frank Burger, Cora Sembach. *Ein Überblick über Verfahren zur rechnergestützten Kooperation*. Stuttgart, 1994.

[cin] *Cinema 4D XL – Referenzhandbuch.*

[Dau95] M. Dauelsberg. *Interaktive multimediale Anwendungen-Seminarvortrag 30.10.1995*. Universität Oldenburg, Fachbereich Informatik, 1995.

[DB95] Stephan Ehrmann Detlef Beyer. Illusion rundherum. *c't*, (8):104, 1995.

[DB97] Marco Schlattmann Dietrich Boles, Peter Dawabi. Antrag auf Durchfuehrung der Projektgruppe Virtuelle naturwissenschaftlich-technische Labore im Internet, 1997. http://www-is.informatik.uni-oldenburg.de/ dibo/teaching/pg-labor/antrag.html.

[DM95] James Hendler Drew McDermott. Planning: What it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence*, 76(1-2), 1995.

[Edi] RenderSoft VRML Editor.
http://homer.pacific.net.sg/ jupboo/.

[FWH97] Aemilian Hron Friedrich W. Hesse, Baerbel Garsoffky. *Information und Lernen mit Multimedia*, sammelwerk Interface-Design fuer computerunterstuetztes kooperatives Lernen. Beltz PsychologieVerlagsUnion, 1997.

[Haa97] Johannis Haack. *Information und Lernen mit Multimedia*, sammelwerk Interaktivitaet als Kennzeichen von Multimedia und Hypermedia. Beltz PsychologieVerlagsUnion, 1997.

[Hag98] Holger Hagedorn. Rundumblick. *computer FOTO*, (4):48, 1998.

[Har87] D. Harel. Statecharts: A visual formalism of complex systems. *Science of Computer Programming 8*, page 231ff, 1987.

[Has97] Hans Lothar Hase. *Dynamische virtuelle Welten mit VRML 2.0*. dpunkt - Verlag für digitale Technologie GmbH, Heidelberg, erste edition, 1997.

[Jac92] Ivar Jacobson. *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, 1992.

[JDF94] Andries van Dam James D. Foley. *Grundlagen der Computergraphik*. Addison-Wesley, 1994.

[KF88] Jochen Ludewig Karol Fruehauf, Helmut Sandmayr. *Software-Projektmanagement und -Qualitätssicherung*. B.G. Teubner Stuttgart Verlag, erste edition, 1988.

[Leu97] Detlev Leutner. *Information und Lernen mit Multimedia*, sammelwerk Adaptivitaet und Adaptierbarkeit multimedialer Lehr- und Lernsysteme. Beltz PsychologieVerlagsUnion, 1997.

[LJI97] Paul Klimsa Ludwig J. Issing. *Information und Lernen mit Multimedia*. Beltz PsychologieVerlagsUnion, Weinheim, 2., ueberarbeitete edition, 1997.

[Lov96] Dr. Jörn Loviscach. Virutelle bühne – maxon cinema 4d. *c't (Magazin für Computertechnik)*, 11, 1996.

[Lov97a] Dr. Jörn Loviscach. Cartoons in 3d – ray dream stuio 5. *c't (Magazin für Computertechnik)*, 10, 1997.

[Lov97b] Dr. Jörn Loviscach. Traumfabriken - aktuelle 3d-software. *c't (Magazin für Computertechnik)*, 1, 1997.

[Oec98] Joachim Oechslin. Top-modell. *c't (Magazin für Computertechnik)*, 6:362–369, 1998.

[Oes97] Bern Oestereich. *Objektorientierte Softwareentwicklung mit der Unified Modeling Language.* R. Oldenbourg Verlag, 1997.

[Old98] Simon Oldeboershuis. Entwicklung eines virtuellen programmierlabors für das www. Master's thesis, Universitaet Oldenburg, 1998.

[PC91a] Ed. Yourdon P. Coad. *Object Oriented Analysis*. Prentice Hall, 1991.

[PC91b] Ed. Yourdon P. Coad. *Object Oriented Design*. Prentice Hall, 1991.

[Per98] Christian Perle. Strahlenförmig. *Linux Magazin – www.linux-magazin.de*, 5 1998.

[PF85] Yumi Iwasaki Peter Friedland. The concept and implementation of skeletal plans. *Automated Reasoning*, I, 1985.

[Pom93] Pomberger/Blaschek. *Software Engineering-Prototyping und objektorientierte Software-Entwicklung*. C.Hanser Verlag, 1993.

[pov] The peristance of vision raytracer. www.povray.org.

[Rie98] Rufus Rieder. Spaßmacher. *Screen Multimedia*, (2):44, 1998.

[RS95] Klara Nahrstedt Ralf Steinmetz. *Multimedia: Computing, Communications and Applications*. Prentice Hall P T R, New Jersey, 1995.

[Rum91] Jim Rumbaugh. *Object Oriented Modeling and Design*. Prentice Hall, 1991.

[Sch95] Henner Schierenbeck. *Grundzüge der Betriebswirtschaftslehre*. Oldenbourg Verlag, zwoelfte edition, 1995.

[Sch96] Tom Schröter. Puppenkiste – autodesk 3d studion max. *c't (Magazin für Computertechnik)*, 5, 1996.

[Sof] Cosmo Software.
http://cosmosoftware.com/.

[Spe98] Tom Sperlich. In 3d um die welt. *c't*, page 48f, 8 1998.

[SS98] S. Mellor S. Shlaer. *Object-Oriented Systems Analysis - Modeling the Worl in Data*. Yourdon Press, 1998.

[Ste81a] Mark Stefik. Planning and meta-planning (MOLGEN: Part 2). *Artificial Intelligence*, 16, 1981.

[Ste81b] Mark Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16, 1981.

[Su95] H. Schmidt u.a. *Der kreative Prozeß*. Screen Multimedia 1/95-11/95, 1995.

[Sup] Superscape.
http://www.3dwebmaster.com/.

[Ter97] Sigmar-Olaf Tergan. *Information und Lernen mit Multimedia*, sammelwerk Hypertext und Hypermedia: Konzeption, Lernmoeglichkeiten, Lernprobleme. Beltz PsychologieVerlagsUnion, 1997.

[VK] VRML-Konsortium.
http://www.vrml.org/.

[Wag94] Sean Wagstaff. *3-D Starter Kit*. Hayden Books, 1994.

[Wei97] Bernd Weidenmann. *Information und Lernen mit Multimedia*, sammelwerk Abbilder in Multimedia-Anwendungen. Beltz PsychologieVerlagsUnion, 1997.